1991-09

An efficient heuristic scheduler for hard real-time systems

Levine, John Glenn

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/26526

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN EFFICIENT HEURISTIC SCHEDULER FOR HARD REAL-TIME SYSTEMS

by

John Glenn Levine

September 1991

Thesis Advisor: Man-Tak Shing
Co-Advisor: LuQi

Approved for public release; distribution is unlimited.

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification UNCLASSIFIED

1b Restrictive Markings

2a Security Classification Authority

2b Declassification/Downgrading Schedule

3 Distribution Availability of Report
Approved for public release; distribution is unlimited.

4 Performing Organization Report Number(s)

5 Monitoring Organization Report Number(s)

6a Name of Performing Organization
Naval Postgraduate School

6b Office Symbol *(If Applicable)*
CS

7a Name of Monitoring Organization
Naval Postgraduate School

6c Address *(city, state, and ZIP code)*
Monterey, CA 93943-5000

7b Address *(city, state, and ZIP code)*
Monterey, CA 93943-5000

8a Name of Funding/Sponsoring Organization
National Science Foundation

8b Office Symbol *(If Applicable)*

9 Procurement Instrument Identification Number
CCR-9058453

8c Address *(city, state, and ZIP code)* 1800 G. St., NW
Washington, DC 20550

10 Source of Funding Numbers

Program Element Number	Project No	Task No	Work Unit Accession No

11 Title *(Include Security Classification)* AN EFFICIENT HEURISTIC SCHEDULER FOR HARD REAL-TIME SYSTEMS

12 Personal Author(s) John Glenn Levine

13a Type of Report
Master's Thesis

13b Time Covered
From To

14 Date of Report *(year, month,day)*
1991, September

15 Page Count
120

16 Supplementary Notation The views expressed in this paper are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes

Field	Group	Subgroup

18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Rapid prototyping; hard real-time systems; simulated annealing

19 Abstract *(continue on reverse if necessary and identify by block number*

The requirement for efficient scheduling algorithms for the development of hard real-time systems resulted in much effort directed toward the development of high performance scheduling algorithms. The algorithms developed up to this point for the Computer Aided Prototyping System (CAPS) do not satisfy the requirements for a efficient static scheduling algorithm. The existing static scheduler neither performs efficiently nor produces correct results for all input cases.

This thesis represents the research conducted to develop a fast heuristic static scheduling algorithm based on the principles of simulated annealing. In addition, this thesis describes the development of new data structures that simplify the static scheduler and maximize system resources. Several of the existing scheduling algorithms were re-implemented to make use of the new data structures and provide correct results. Any feasible schedule produced by these scheduling algorithms guarantees that both timing and precedence constraints are met. The primary goal of this thesis was to produce an efficient and effective scheduler to support the CAPS system.

20 Distribution/Availability of Abstract
☒ unclassified/unlimited □ same as report □ DTIC users

21 Abstract Security Classification
Unclassified

22a Name of Responsible Individual
M. Shing

22b Telephone *(Include Area code)*
(408) 646-2634

22c Office Symbol
CS/Sh

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page
Unclassified

AN EFFICIENT HEURISTIC SCHEDULER FOR HARD REAL-TIME SYSTEMS

by
John Glenn Levine
Cpt, U.S. Army
B.S., U.S.M.A., 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
26 September 1991

ABSTRACT

The requirement for efficient scheduling algorithms for the development of hard real-time systems resulted in much effort directed toward the development of high performance scheduling algorithms. The algorithms developed up to this point for the Computer Aided Prototyping System (CAPS) do not satisfy the requirements for a efficient static scheduling algorithm. The existing static scheduler neither performs efficiently nor produces correct results for all input cases.

This thesis represents the research conducted to develop a fast heuristic static scheduling algorithm based on the principles of simulated annealing. In addition, this thesis describes the development of new data structures that simplify the static scheduler and maximize system resources. Several of the existing scheduling algorithms were re-implemented to make use of the new data structures and provide correct results. Any feasible schedule produced by these scheduling algorithms guarantees that both timing and precedence constraints are met. The primary goal of this thesis was to produce an efficient and effective scheduler to support the CAPS system.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND: HARD REAL-TIME SYSTEMS

Large scale hard real-time systems are important to both civilian and military operations. Hard real-time systems are defined as those systems in which the correctness of the system depends not only on the logical results of the computation, but also on the time at which the results are produced. If results are not produced in a timely manner, disastrous results may occur. Examples of hard real-time systems include air traffic control systems, telecommunications systems, space shuttle control avionics systems, C^3I systems, and future Strategic Defense Initiative (SDI) systems. Most hard real-time systems are specialized and complex, require a high degree of fault tolerance, and are typically embedded in a larger system. To overcome the complexity in the design and development of such systems, software engineers now use a new approach, called rapid prototyping, to build and maintain these systems. Rapid prototyping is a means for stabilizing and validating the requirements for complex systems (e.g. embedded control systems with hard real-time constraints) by helping the customer visualize system behavior prior to detailed implementation. The Computer Aided Prototyping System (CAPS), which is being developed at the Naval Postgraduate School, supports an iterative prototyping process characterized by exploratory design and extensive prototype evolution, thus enabling the engineers to produce complex systems that match user needs and reduce the need for expensive modifications after delivery.

B. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

CAPS consists of several modules. Figure 1 below describes the major software modules of CAPS. The user interface consists largely of a graphical editor for the formal prototyping language called Prototyping System Description Language (PSDL). Future implementations of this module will also have a syntax directed editor. The second module is the Software Database System which includes the Rewrite Subsystems, the Software Design Management Subsystem, and the Reusable Software Component Database. The third module is the Execution Support System (ESS). This module contains the PSDL Translator, the Static Scheduler, and the Dynamic Scheduler. Figure 2 shows the implementation and interfaces of the ESS. This thesis is concerned with the static scheduler component of the ESS.

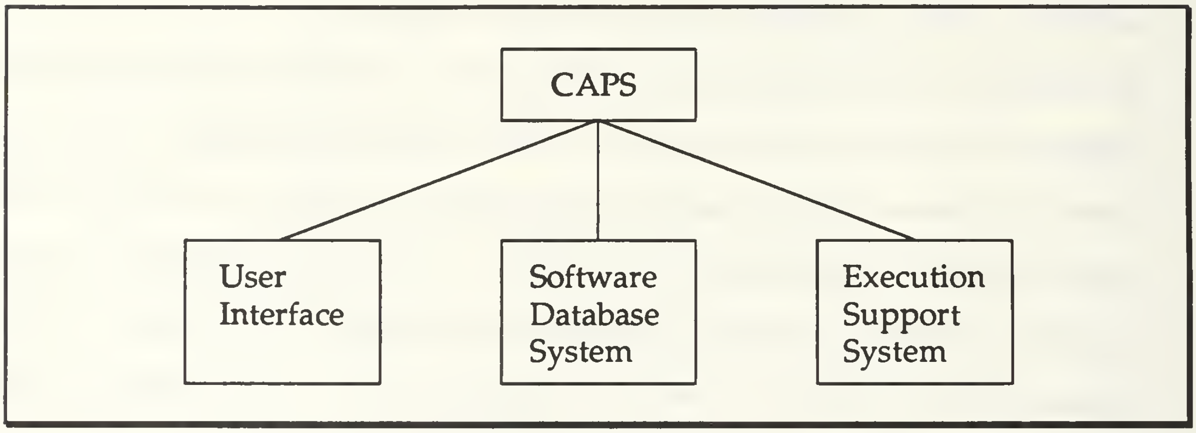


Figure 1. Major Software Tools of CAPS

The Dynamic Scheduler acts as a run-time executive when exercising the system. It schedules operators without timing constraints, which are not include in the static schedule, by using spare capacity in the static schedule. It

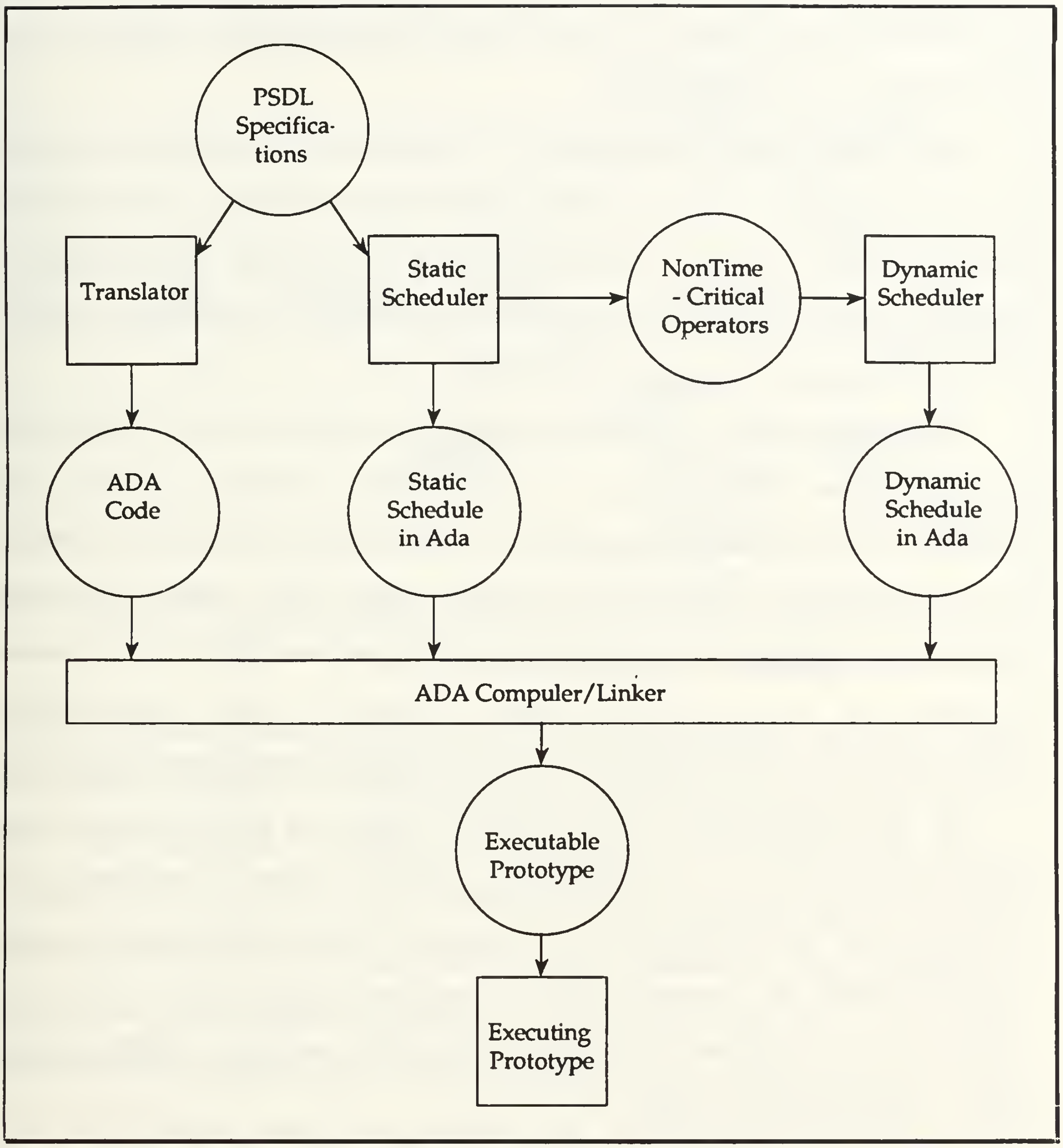


Figure 2. The Execution Support System

handles run-time exceptions and hardware/operator interrupts. It communicates with the user interface during prototype runs. Thus, it performs like a miniature operating system. While the problems involved in this subsystem are interesting, it is the static scheduler that deals with the issues addressed in this proposal.

The purpose of the static scheduler is to build a static schedule for a set of tasks that must obey both precedence and timing constraints. This schedule gives the order of execution and the timing of the operators. The schedule is legal and feasible if both the precedence relationships are maintained and the timing constraints are guaranteed to be met.

The existing static scheduler is described in (Janson, 1988), (Killic 1989) and (Cervantes, 1988). Figure 3 is a data flow description of the static scheduler. The following paragraphs are a description of the static scheduler that was originally implemented by (Janson, 1988), (Killic, 1989), (Cervantes, 1988) and modified by the work described in this thesis. The Static Scheduler consists of five modules—PSDL_READER, FILE_PROCESSOR, TOPOLOGICAL_SORTER, HARMONIC_BLOCK_BUILDER, and OPERATOR_SCHEDULER.

The first component, PSDL_READER, reads and processes the PSDL prototyping program. It is essentially a filter that removes information not needed by the Static Scheduler. The output of this module is the text file ATOMIC.INFO that contains all the operators along with any timing constraints the operators may have and the link statements which describe PSDL implementation graphs.

The second component, FILE_PROCESSOR, analyzes the text file generated by the PSDL_READER and separates the information into a linked list data structure called THE_GRAPH and a file called NON_CRITS. It then

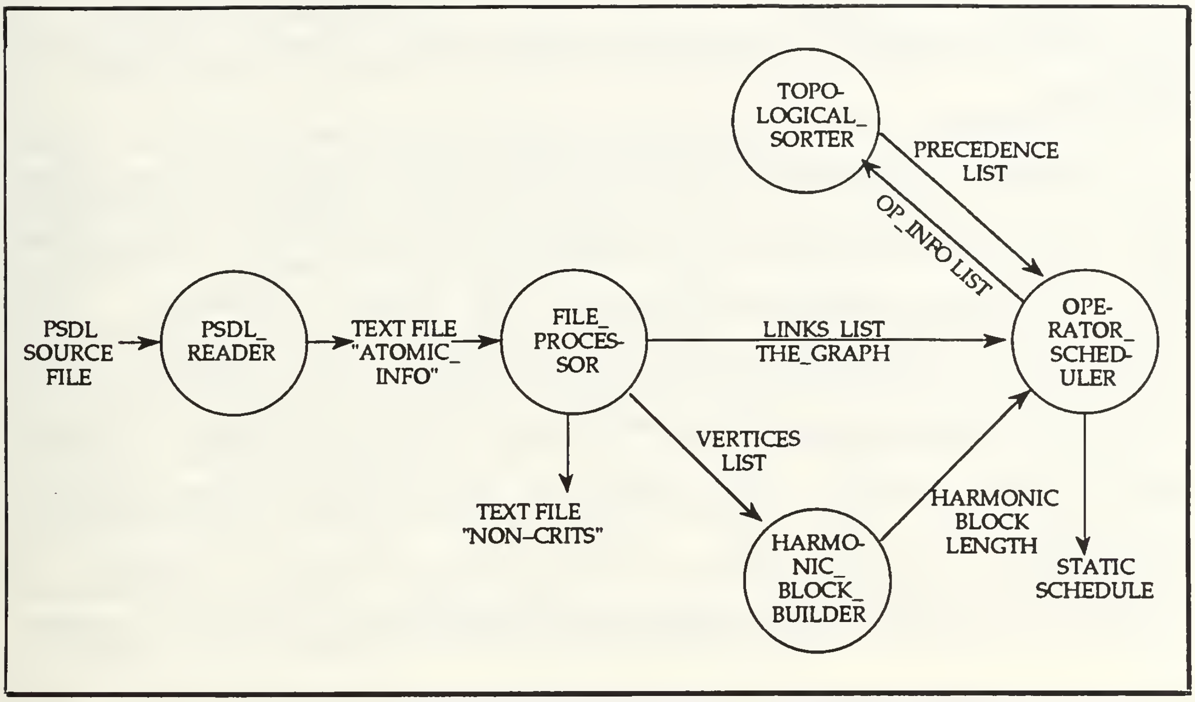


Figure 3. Static Scheduler Data Flow Diagram

converts sporadic operators into their periodic equivalents. The information is separated based on its destination and the additional processing required. THE_GRAPH, which is a graph structure, as indicated in Figure 4 below contains two linked lists. The "VERTICES" list contains a list of all time-critical operators and their associated timing constraints. The "LINKS" list contains the link statements which are a syntactical description of the PSDL implementation graphs and indicates the data flows between operators. The "VERTICES" list is used by the HARMONIC_BLOCK_BUILDER module and the "LINKS" list is used by the OPERATOR_SCHEDULER to develop a OP_INFO list. The OP_INFO list is then used by the TOPOLOGICAL_SORTER to develop a precedence list for the operators to be scheduled. The entire structure THE_GRAPH is also used by

OPERATOR_SCHEDULER to develop a static schedule. The NON_CRITS file contains a list of all non-critical operators that is used by the Dynamic Scheduler.

The third component, TOPOLOGICAL_SORTER, performs a topological sort on the OP_INFO data structure. Using the OP_INFO list is a change from the previous implementations of the Static Scheduler. The TOPOLOGICAL_SORTER has also been rewritten. It now develops a true topological ordering and is not dependent on a specific ordering of operators in the PSDL input file. The result is a total ordering of the operators depending on data flow. This total ordering is passed to OPERATOR_SCHEDULER module as the PRECEDENCE_LIST data structure.

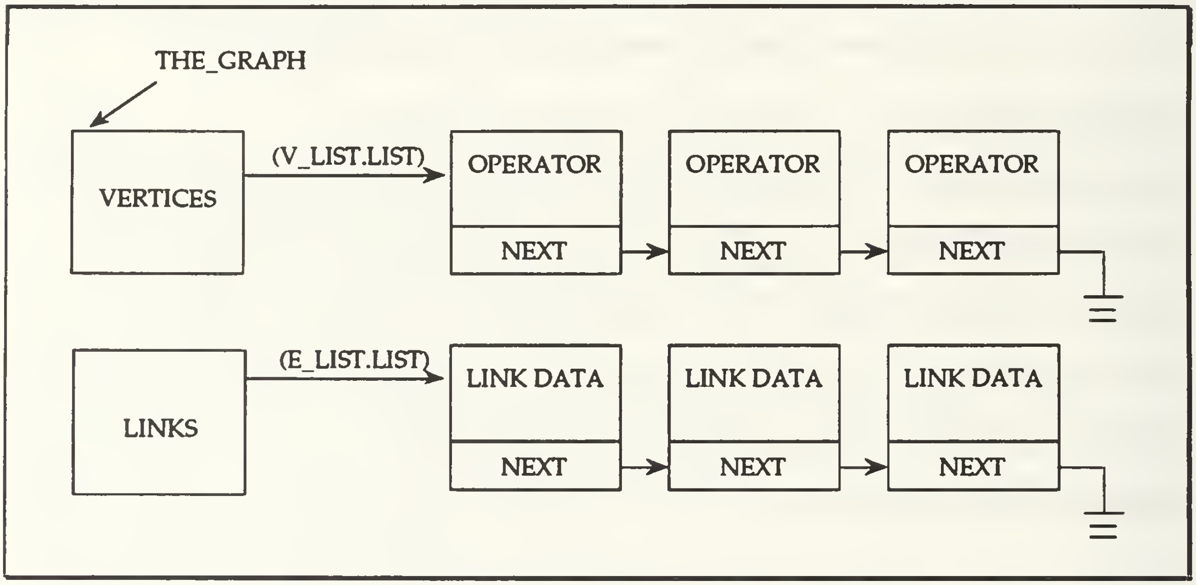


Figure 4. Graphical Representation of THE_GRAPH Linked List Structure

The fourth component, HARMONIC_BLOCK_BUILDER determines the Harmonic Block Length of the static schedule to be developed. A harmonic

C. PERIODIC OPERATORS

This section is based upon the background work done in (Cervantes, 1989). Periodic operators are triggered by temporal events and must occur at regular time intervals. The timing constraints of each periodic operator OP_i consists of a specific period period(OP_i), a maximum execution time MET(OP_i), and a deadline finish_within(OP_i). Denote the kth instance of OP_i by $OP_{i,k}$, the start time of $OP_{i,k}$ by start_time($OP_{i,k}$), and the completion time of $OP_{i,k}$ by completion($OP_{i,k}$). For k > 1, define earliest_start_time($OP_{i,k}$), the earliest starting time of $OP_{i,k}$, as start_time($OP_{i,1}$) + (k-1) * period(OP_i) and deadline($OP_{i,k}$), the latest completion time of $OP_{i,k}$, as earliest_start_time($OP_{i,k}$) + finish_within(OP_i). Then

$$\text{start\_time}(OP_{i,k}) >= \text{earliest\_start\_time}(OP_{i,k})$$

and

$$\text{start\_time}(OP_{i,k}) + \text{MET}(OP_i) <= \text{deadline}(OP_{i,k}).$$

The precedence constraints among a given set of operators are specified in the form of a directed acyclic graph G. The precedence constraints are defined by the communications among the operators that compose the system being developed. PSDL operators communicate by means of named data streams. All data values carried by a data stream must be instances of a specific abstract data type associated with the stream. There are two different types of data streams in PSDL, dataflow streams and sampled streams. Dataflow streams are used in applications where the values in the stream must not be lost or replicated and the period of the producer and consumer of the data must be the same (lockstep performance). Sampled streams are used in applications

C. PERIODIC OPERATORS

This section is based upon the background work done in (Cervantes, 1989). Periodic operators are triggered by temporal events and must occur at regular time intervals. The timing constraints of each periodic operator OP_i consists of a specific period period(OP_i), a maximum execution time MET(OP_i), and a deadline finish_within(OP_i). Denote the kth instance of OP_i by $OP_{i,k}$, the start time of $OP_{i,k}$ by start_time($OP_{i,k}$), and the completion time of $OP_{i,k}$ by completion($OP_{i,k}$). For k > 1, define earliest_start_time($OP_{i,k}$), the earliest starting time of $OP_{i,k}$, as start_time($OP_{i,1}$) + (k-1) * period(OP_i) and deadline($OP_{i,k}$), the latest completion time of $OP_{i,k}$, as earliest_start_time($OP_{i,k}$) + finish_within(OP_i). Then

$$\text{start\_time}(OP_{i,k}) >= \text{earliest\_start\_time}(OP_{i,k})$$

and

$$\text{start\_time}(OP_{i,k}) + \text{MET}(OP_i) <= \text{deadline}(OP_{i,k}).$$

The precedence constraints among a given set of operators are specified in the form of a directed acyclic graph G. The precedence constraints are defined by the communications among the operators that compose the system being developed. PSDL operators communicate by means of named data streams. All data values carried by a data stream must be instances of a specific abstract data type associated with the stream. There are two different types of data streams in PSDL, dataflow streams and sampled streams. Dataflow streams are used in applications where the values in the stream must not be lost or replicated and the period of the producer and consumer of the data must be the same (lockstep performance). Sampled streams are used in applications

D. ORGANIZATION

The objective of this thesis is to describe a new heuristic static scheduling algorithm that uses the principles of simulated annealing to develop a feasible schedule if one exists. To do so this thesis is organized as follows: Chapter II describes the static scheduling algorithms that exist in CAPS for a single processor environment; Chapter III is a description of the new heuristic scheduling algorithm. It includes a description of the simulated annealing process and the implementation of this process in the static scheduler; Chapter IV is a description of the new data structure and modifications made to existing modules that improve the performance of the static scheduler; Chapter V is an evaluation of this new algorithm; and Chapter VI presents conclusions and recommendations for future work.

D. ORGANIZATION

The objective of this thesis is to describe a new heuristic static scheduling algorithm that uses the principles of simulated annealing to develop a feasible schedule if one exists. To do so this thesis is organized as follows: Chapter II describes the static scheduling algorithms that exist in CAPS for a single processor environment; Chapter III is a description of the new heuristic scheduling algorithm. It includes a description of the simulated annealing process and the implementation of this process in the static scheduler; Chapter IV is a description of the new data structure and modifications made to existing modules that improve the performance of the static scheduler; Chapter V is an evaluation of this new algorithm; and Chapter VI presents conclusions and recommendations for future work.

addressed in (Janson, 1988). This chapter examines the five scheduling algorithms currently implemented in CAPS. These five algorithms are Harmonic Block with Precedence Constraints, Earliest Start, Earliest Deadline, Branch and Bound, and Exhaustive Enumeration. The first three algorithms were described in detail in (Kilic, 1989), and the remaining two were described in detail in (Fan, 1990).

B. HARMONIC BLOCK WITH PRECEDENCE CONSTRAINTS

This algorithm attempts to find a feasible schedule by scheduling the operators in the order that they appear in a topological ordering. If any of the operators violate a timing constraint, the schedule being developed is rejected. Since in most hard real-time systems there exists more than one topological ordering of operators there are cases where one ordering will produce a feasible schedule while another will not. This algorithm does not adjust the topological ordering in order to find a feasible schedule.

C EARLIEST START TIME SCHEDULING ALGORITHM

In the original algorithm (Bra, 1971), each transaction must have an earliest start time. That is, each transaction becomes available at time a_i, must be completed by b_i, and requires c_i units of time. Pre-emption of transactions is allowed in this algorithm but transaction precedence is normally not allowed. The version of the algorithm that is implemented in CAPS allows precedence but does not allow pre-emption. Transactions are scheduled in this algorithm based on the system clock, the earliest start time of a transaction, and the priority of the transaction. The algorithm assigns a time slot to the newest transaction based on its earliest start time. If two or more addressed in (Janson, 1988). This chapter examines the five scheduling algorithms currently implemented in CAPS. These five algorithms are Harmonic Block with Precedence Constraints, Earliest Start, Earliest Deadline, Branch and Bound, and Exhaustive Enumeration. The first three algorithms were described in detail in (Kilic, 1989), and the remaining two were described in detail in (Fan, 1990).

B. HARMONIC BLOCK WITH PRECEDENCE CONSTRAINTS

This algorithm attempts to find a feasible schedule by scheduling the operators in the order that they appear in a topological ordering. If any of the operators violate a timing constraint, the schedule being developed is rejected. Since in most hard real-time systems there exists more than one topological ordering of operators there are cases where one ordering will produce a feasible schedule while another will not. This algorithm does not adjust the topological ordering in order to find a feasible schedule.

C EARLIEST START TIME SCHEDULING ALGORITHM

In the original algorithm (Bra, 1971), each transaction must have an earliest start time. That is, each transaction becomes available at time a_i, must be completed by b_i, and requires c_i units of time. Pre-emption of transactions is allowed in this algorithm but transaction precedence is normally not allowed. The version of the algorithm that is implemented in CAPS allows precedence but does not allow pre-emption. Transactions are scheduled in this algorithm based on the system clock, the earliest start time of a transaction, and the priority of the transaction. The algorithm assigns a time slot to the newest transaction based on its earliest start time. If two or more

E. THE NEED FOR A NEW SCHEDULING ALGORITHM

There is a gap in the current static scheduler. Three algorithms exist that attempt to develop a quick solution. These algorithms only find feasible solutions for very simple hard real-time systems but fail to find a feasible solution as systems become more complex. Exhaustive Enumeration and Branch and Bound, on the other hand, will find a feasible schedule if such a schedule exists, but both are very costly due to their time complexity.

There exists a need for a fast algorithm that is capable of producing a feasible solution. The proposed heuristic algorithm, which is based on the simulated annealing approach, appears to be the best compromise between simple-minded and exponential time algorithms already implemented in CAPS.

F. SUMMARY

This chapter presented a sample of previous algorithms developed to solve the real-time scheduling requirement. These algorithms have inherent weaknesses such as an inability to handle complex topological orderings that do not immediately produce solutions or they have a high degree of time complexity. Since the static scheduling problem is NP-hard (Zdrzalka, 1988), systemic global search is the only guaranteed way to return a feasible static schedule for a hard real-time system if such a schedule exists. The exhaustive enumeration algorithm has already been implemented in CAPS to accomplish this. This algorithm has demonstrated to be very costly in practice.

E. THE NEED FOR A NEW SCHEDULING ALGORITHM

There is a gap in the current static scheduler. Three algorithms exist that attempt to develop a quick solution. These algorithms only find feasible solutions for very simple hard real-time systems but fail to find a feasible solution as systems become more complex. Exhaustive Enumeration and Branch and Bound, on the other hand, will find a feasible schedule if such a schedule exists, but both are very costly due to their time complexity.

There exists a need for a fast algorithm that is capable of producing a feasible solution. The proposed heuristic algorithm, which is based on the simulated annealing approach, appears to be the best compromise between simple-minded and exponential time algorithms already implemented in CAPS.

F. SUMMARY

This chapter presented a sample of previous algorithms developed to solve the real-time scheduling requirement. These algorithms have inherent weaknesses such as an inability to handle complex topological orderings that do not immediately produce solutions or they have a high degree of time complexity. Since the static scheduling problem is NP-hard (Zdrzalka, 1988), systemic global search is the only guaranteed way to return a feasible static schedule for a hard real-time system if such a schedule exists. The exhaustive enumeration algorithm has already been implemented in CAPS to accomplish this. This algorithm has demonstrated to be very costly in practice.

III. DESCRIPTION OF THE ALGORITHM TO HANDLE THE HARD REAL-TIME SCHEDULING PROBLEM

A. SIMULATED ANNEALING

The use of simulated annealing to solve combinatorial optimization problems is an area that has received much attention lately. Combinatorial optimization problems are those whose configuration of elements are finite or countably infinite. An example combinatorial optimization problem is the assignment problem where there are a number of personnel available to do an equal number of jobs. The cost for each person to do each job is known. The goal is to assign each person to a job so that the total cost is as small as possible (Otten, 1989). There are a wide range of combinatorial optimization problems that the simulated annealing approach can be utilized for. These include graph partitioning, graph coloring, number partitioning, VLSI design, and travelling salesman type problems.

Simulated annealing is based on the behavior of physical systems and the laws of thermodynamics. The way that liquids freeze and crystalize or metals cool and anneal are the principles upon which simulated annealing is based. At high temperature, liquid molecules move freely with respect to one another. As the liquid cools, this mobility is lost. Atoms line up and form a pure crystal that is at a minimum energy level. As the system cools slowly nature finds the minimum energy state (Flannery, 1984). Examining simulated annealing in non-physical terms, a comparison is made to the concept of local optimization or iterative improvement. Local optimization

III. DESCRIPTION OF THE ALGORITHM TO HANDLE THE HARD REAL-TIME SCHEDULING PROBLEM

A. SIMULATED ANNEALING

The use of simulated annealing to solve combinatorial optimization problems is an area that has received much attention lately. Combinatorial optimization problems are those whose configuration of elements are finite or countably infinite. An example combinatorial optimization problem is the assignment problem where there are a number of personnel available to do an equal number of jobs. The cost for each person to do each job is known. The goal is to assign each person to a job so that the total cost is as small as possible (Otten, 1989). There are a wide range of combinatorial optimization problems that the simulated annealing approach can be utilized for. These include graph partitioning, graph coloring, number partitioning, VLSI design, and travelling salesman type problems.

Simulated annealing is based on the behavior of physical systems and the laws of thermodynamics. The way that liquids freeze and crystalize or metals cool and anneal are the principles upon which simulated annealing is based. At high temperature, liquid molecules move freely with respect to one another. As the liquid cools, this mobility is lost. Atoms line up and form a pure crystal that is at a minimum energy level. As the system cools slowly nature finds the minimum energy state (Flannery, 1984). Examining simulated annealing in non-physical terms, a comparison is made to the concept of local optimization or iterative improvement. Local optimization energy states the probability for making an uphill move still exists. As indicated in Figure 5 above, uphill moves allow the algorithm to leave a poor local solution (point A or point B) and reach a better solution in the vicinity of point C. This general scheme of always taking a downhill step while occasionally taking an uphill step is known as the Metroplis algorithm, named after Metroplis, the scientist, who with his coworkers first investigated simulated annealing in 1953 (Press, 1984).

The choice of a probability function to determine if an uphill movement is allowed is an important consideration. At each step of the simulated annealing algorithm a new state is constructed based on the current state. This new state is constructed by randomly displacing or adjusting a randomly selected element. If this new state has a lower cost than the current state, the new state is accepted as the current state. If the new state has a higher cost than the current state, the new state is accepted with the probability:

$$\exp(-\Delta e / kT).$$

This probability function is known as the Boltzman probability distribution where:

Δe = difference in cost between new state and current state

k = Boltzman's constant of nature relating temperature to energy

T = Current Temperature

A characteristic of this probability function is that at very high temperatures every new state has an almost even chance of being accepted as the current state. At low temperatures the states with a lower cost have a higher probability of being accepted as the current state.

energy states the probability for making an uphill move still exists. As indicated in Figure 5 above, uphill moves allow the algorithm to leave a poor local solution (point A or point B) and reach a better solution in the vicinity of point C. This general scheme of always taking a downhill step while occasionally taking an uphill step is known as the Metroplis algorithm, named after Metroplis, the scientist, who with his coworkers first investigated simulated annealing in 1953 (Press, 1984).

The choice of a probability function to determine if an uphill movement is allowed is an important consideration. At each step of the simulated annealing algorithm a new state is constructed based on the current state. This new state is constructed by randomly displacing or adjusting a randomly selected element. If this new state has a lower cost than the current state, the new state is accepted as the current state. If the new state has a higher cost than the current state, the new state is accepted with the probability:

$$\exp(-\Delta e/kT).$$

This probability function is known as the Boltzman probability distribution where:

Δe = difference in cost between new state and current state

k = Boltzman's constant of nature relating temperature to energy

T = Current Temperature

A characteristic of this probability function is that at very high temperatures every new state has an almost even chance of being accepted as the current state. At low temperatures the states with a lower cost have a higher probability of being accepted as the current state.

The annealing schedule sets after how many random changes in the configuration is each downward step in T taken, and how large that step is. The range of the annealing temperature and the value of the annealing schedule are normally established from trial and error experimentation (Flannery, 1984).

A pseudocode representation of the simulated annealing algorithm based on the algorithm proposed in (Johnson, 1989) follows:

```
BEGIN
      GET AN INITIAL SOLUTION
      SET INITIAL TEMPERATURE $T > 0$
      WHILE $T > 0$ LOOP
            FOR I IN 1..L LOOP
                  GENERATE A NEW SOLUTION
                  $\Delta e$ = E(NEW SOLUTION) - E(CURRENT SOLUTION)
                  IF $\Delta e <= 0$ THEN
                        CURRENT SOLUTION := NEW SOLUTION
                  ELSE
                        CURRENT SOLUTION := NEW SOLUTION
                              WITH PROBABILITY $\exp(-\Delta e/T)$
                  END IF
            END LOOP
            ADJUST TEMPERATURE ($T = rT$)
      END LOOP
END
      WHERE       $T$ = TEMPERATURE
                  $r$ = COOLING FACTOR
                  $L$ = NUMBER OF TRIALS TO PERFORM AT EACH TEMPERATURE
                  $\Delta e$= DIFFERENCE IN COSTS BETWEEN TWO SOLUTIONS
```

The choice of values for T, r, and L have a significant impact on the annealing schedule. The higher the initial temperature, the higher the cooling factor, and the larger the number of trials at each temperature result in more solutions being examined in order to find an optimum solution. The goal in choosing these parameters is to pick them so that a sufficient but not excessive number of solutions are examined. These values are normally chosen arbitrarily and adjusted through experimentation. The next section of

The annealing schedule sets after how many random changes in the configuration is each downward step in T taken, and how large that step is. The range of the annealing temperature and the value of the annealing schedule are normally established from trial and error experimentation (Flannery, 1984).

A pseudocode representation of the simulated annealing algorithm based on the algorithm proposed in (Johnson, 1989) follows:

```
BEGIN
      GET AN INITIAL SOLUTION
      SET INITIAL TEMPERATURE T > 0
      WHILE T > 0 LOOP
            FOR I IN 1..L LOOP
                  GENERATE A NEW SOLUTION
                  Δe = E(NEW SOLUTION) - E(CURRENT SOLUTION)
                  IF Δe <= 0 THEN
                        CURRENT SOLUTION := NEW SOLUTION
                  ELSE
                        CURRENT SOLUTION := NEW SOLUTION
                              WITH PROBABILITY exp(-Δe/T)
                  END IF
            END LOOP
            ADJUST TEMPERATURE (T = rT)
      END LOOP
END
      WHERE       T = TEMPERATURE
                  r = COOLING FACTOR
                  L = NUMBER OF TRIALS TO PERFORM AT EACH TEMPERATURE
                  Δe= DIFFERENCE IN COSTS BETWEEN TWO SOLUTIONS
```

The choice of values for T, r, and L have a significant impact on the annealing schedule. The higher the initial temperature, the higher the cooling factor, and the larger the number of trials at each temperature result in more solutions being examined in order to find an optimum solution. The goal in choosing these parameters is to pick them so that a sufficient but not excessive number of solutions are examined. These values are normally chosen arbitrarily and adjusted through experimentation. The next section of

The goal of the hard real-time scheduler is to find a feasible schedule for the operators, not the optimum schedule. This means that simulated annealing can be terminated as soon as a feasible schedule is found. Both loops of the annealing algorithm are modified so that if a feasible schedule is found, the loop condition for both loops is satisfied and annealing is terminated. This means that when each iterative solution is tested, it is examined to see if it is a feasible solution. The next section describes what a feasible solution is. If the current schedule is feasible, boolean flags are set so that both loop conditions of the algorithm are satisfied.

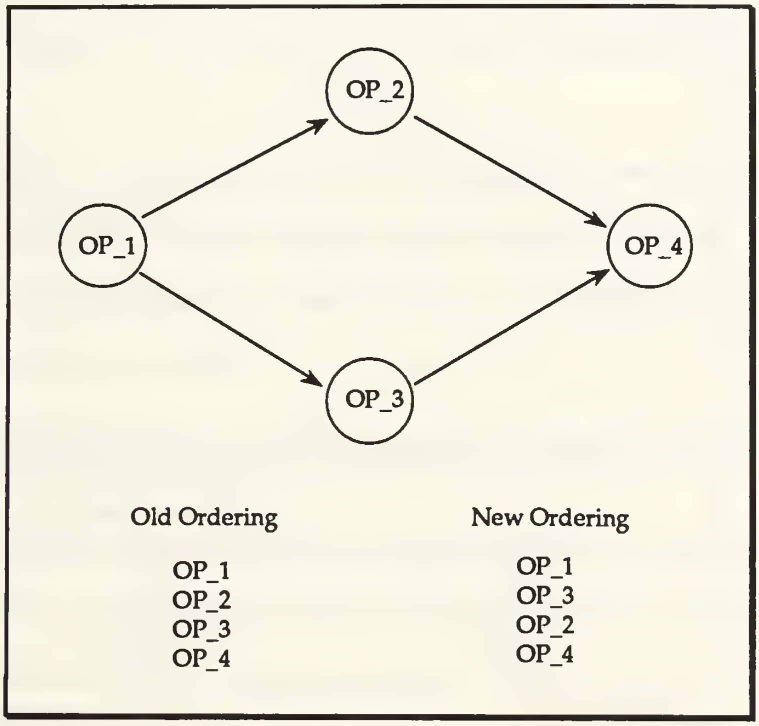


Figure 6. Reordering of Operators Preserving Precedence

The goal of the hard real-time scheduler is to find a feasible schedule for the operators, not the optimum schedule. This means that simulated annealing can be terminated as soon as a feasible schedule is found. Both loops of the annealing algorithm are modified so that if a feasible schedule is found, the loop condition for both loops is satisfied and annealing is terminated. This means that when each iterative solution is tested, it is examined to see if it is a feasible solution. The next section describes what a feasible solution is. If the current schedule is feasible, boolean flags are set so that both loop conditions of the algorithm are satisfied.

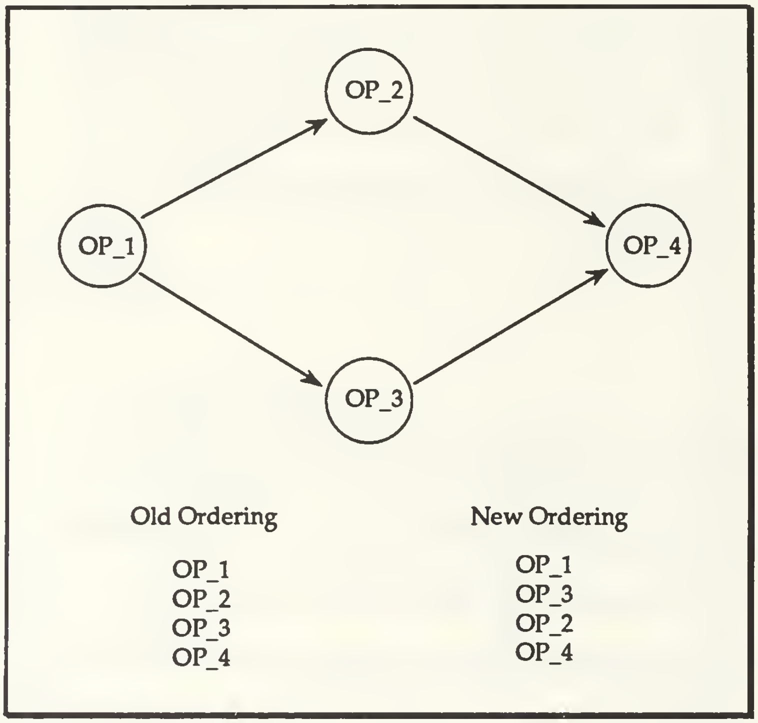


Figure 6. Reordering of Operators Preserving Precedence

The proposed schedule must also be examined to check that the finish time of the last operator in the schedule does not exceed the harmonic block length. The concept of harmonic block length is covered in (Kilic, 1989). The basic idea is that a schedule is developed to fit inside a harmonic block. The length of the harmonic block is the greatest common multiple of the periods of all operators to be scheduled. Once a schedule is developed that fits within the harmonic block, subsequent copies of the block can be made to maintain the hard real-time schedule. Each proposed schedule is examined to insure that the schedule does not exceed the harmonic block length. If a schedule does exceed the harmonic block length, the schedule is not valid since subsequent copies of the schedule will violate the hard real-time timing constraints.

If a schedule is a examined and all timing constraints are satisfied and the harmonic block length is not violated then a feasible schedule exists. At this point the simulated annealing algorithm is terminated and the feasible schedule is returned to CAPS.

E. METHOD FOR PRODUCING A FEASIBLE SCHEDULE FOR A PROPOSED REAL-TIME SYSTEM

The simulated annealing algorithm uses a step by step method to find a feasible solution. These steps include developing an initial solution, testing the initial and subsequent solutions, and adjusting the solution while guaranteeing that operator precedence is maintained. The simulated annealing algorithm is a heuristic (or approximate) approach to solving the scheduling problem for hard real-time systems. It does not guarantee to find a valid solution even if one exists. The goal of this thesis is to develop an

The proposed schedule must also be examined to check that the finish time of the last operator in the schedule does not exceed the harmonic block length. The concept of harmonic block length is covered in (Kilic, 1989). The basic idea is that a schedule is developed to fit inside a harmonic block. The length of the harmonic block is the greatest common multiple of the periods of all operators to be scheduled. Once a schedule is developed that fits within the harmonic block, subsequent copies of the block can be made to maintain the hard real-time schedule. Each proposed schedule is examined to insure that the schedule does not exceed the harmonic block length. If a schedule does exceed the harmonic block length, the schedule is not valid since subsequent copies of the schedule will violate the hard real-time timing constraints.

If a schedule is a examined and all timing constraints are satisfied and the harmonic block length is not violated then a feasible schedule exists. At this point the simulated annealing algorithm is terminated and the feasible schedule is returned to CAPS.

E. METHOD FOR PRODUCING A FEASIBLE SCHEDULE FOR A PROPOSED REAL-TIME SYSTEM

The simulated annealing algorithm uses a step by step method to find a feasible solution. These steps include developing an initial solution, testing the initial and subsequent solutions, and adjusting the solution while guaranteeing that operator precedence is maintained. The simulated annealing algorithm is a heuristic (or approximate) approach to solving the scheduling problem for hard real-time systems. It does not guarantee to find a valid solution even if one exists. The goal of this thesis is to develop an schedule as possible while maintaining precedence. Figure 8 demonstrates the annealing that occurs. Each operator ahead of the operator in question is examined to determine if it is a parent of the operator that violated its timing constraints. The operator continues to move up the schedule until we come to its parent. At this point we insert the operator in question after its parent. Each operator in the new schedule begins at its lower bound or immediately after the preceding operator, which ever is greater. This new schedule is then examined to determine what its cost is and if it is in fact a feasible schedule.

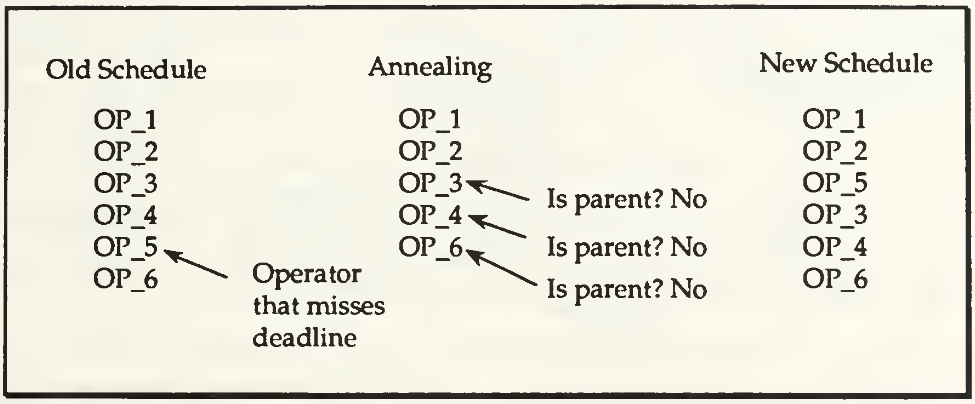


Figure 8. Use of Annealing to Modify a Schedule

If the new schedule has a positive cost that is lower than that of the current schedule, this new schedule is adopted and annealing continues. If the new schedule is costlier than the current schedule, a random choice is made whether to accept the new schedule with its higher cost of keep the current schedule. This choice is made in accordance with the annealing function, which takes into account the current temperature of the system and the difference in cost between the current solution and the new solution. The choice of accepting the new solution with a higher cost over the current schedule as possible while maintaining precedence. Figure 8 demonstrates the annealing that occurs. Each operator ahead of the operator in question is examined to determine if it is a parent of the operator that violated its timing constraints. The operator continues to move up the schedule until we come to its parent. At this point we insert the operator in question after its parent. Each operator in the new schedule begins at its lower bound or immediately after the preceding operator, which ever is greater. This new schedule is then examined to determine what its cost is and if it is in fact a feasible schedule.

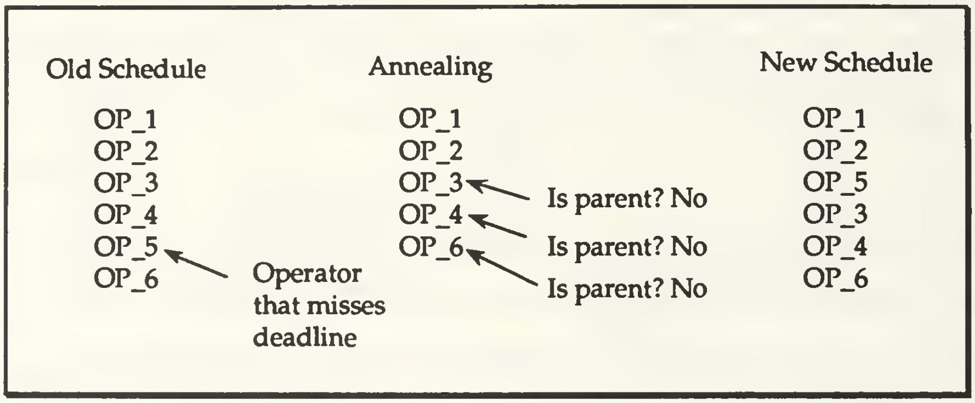


Figure 8. Use of Annealing to Modify a Schedule

If the new schedule has a positive cost that is lower than that of the current schedule, this new schedule is adopted and annealing continues. If the new schedule is costlier than the current schedule, a random choice is made whether to accept the new schedule with its higher cost of keep the current schedule. This choice is made in accordance with the annealing function, which takes into account the current temperature of the system and the difference in cost between the current solution and the new solution. The choice of accepting the new solution with a higher cost over the current

IV. IMPLEMENTATION OF THE STATIC SCHEDULER

A. OBSERVATIONS

The previous implementation of the static scheduler, although functional, does not perform scheduling in the most efficient manner, nor does it handle all types of input. During the development of the new scheduling algorithm problems were identified and corrected in several of the existing packages, which are part of the static scheduler. Development of more efficient data structures resulted in faster execution of all scheduling algorithms and eliminated the requirement for cumbersome input/output between the various components of the static scheduler.

The modification of existing packages and the development of new data structures greatly improved the performance of the new static scheduler while increasing modularity and simplifying the code of the various scheduling algorithms. The implementation of additional scheduling algorithms in the future will become a simpler task because of the work done in this thesis.

B. MODIFICATIONS TO EXISTING PACKAGES

Four packages that made up the static scheduler underwent modification in order to correct errors, increase functionality and improve performance. These four packages are the generic package SEQUENCES, which contained all the linked list routines, the TOPOLOGICAL_SORTER package, the FILE_PROCESSOR package, and the FILES package, which contained all of the global variables and data structures used by the static scheduler.

IV. IMPLEMENTATION OF THE STATIC SCHEDULER

A. OBSERVATIONS

The previous implementation of the static scheduler, although functional, does not perform scheduling in the most efficient manner, nor does it handle all types of input. During the development of the new scheduling algorithm problems were identified and corrected in several of the existing packages, which are part of the static scheduler. Development of more efficient data structures resulted in faster execution of all scheduling algorithms and eliminated the requirement for cumbersome input/output between the various components of the static scheduler.

The modification of existing packages and the development of new data structures greatly improved the performance of the new static scheduler while increasing modularity and simplifying the code of the various scheduling algorithms. The implementation of additional scheduling algorithms in the future will become a simpler task because of the work done in this thesis.

B. MODIFICATIONS TO EXISTING PACKAGES

Four packages that made up the static scheduler underwent modification in order to correct errors, increase functionality and improve performance. These four packages are the generic package SEQUENCES, which contained all the linked list routines, the TOPOLOGICAL_SORTER package, the FILE_PROCESSOR package, and the FILES package, which contained all of the global variables and data structures used by the static scheduler.

traverses a linked list freeing each node in that list. The second COPY_LIST, allows the contents of one list to be copied into another list. This procedure will work with lists of the same or different lengths. The need for these two routines to improve memory management came about as a result of the development of the simulated annealing algorithm. Since this algorithm repeatedly generates new schedules, a computer system's memory would rapidly fill to capacity if discarded schedules were not reclaimed for their memory.

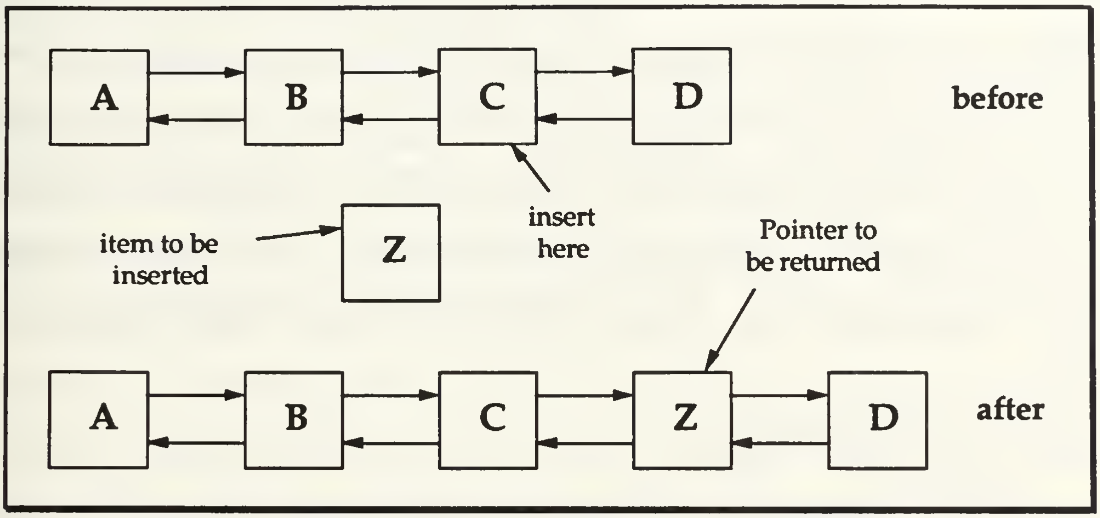


Figure 10. Effect of the INSERT_NEXT Linked List Routine

2. TOPOLOGICAL_SORTER

The original topological sorter only worked when the input was received in a certain order. True topological orderings were not found. This sorter did not handle cases of multiple data links between operators. The sorter also required numerous traversals of various linked lists in order to accomplish a topological ordering of operators.

The new TOPOLOGICAL_SORTER (T_SORT) is a simpler and faster implementation of the topological ordering algorithm. It uses an array that is initialized to the in-degree of each operator. The new scheduler always augments the given precedence graph with a dummy start node. This dummy start node has in-degree zero and is connected to all the operators with in-degree zero in the original precedence graph. The dummy start node is the only operator in the queue of operators to be processed initially. We remove the operator v from the head of the queue and place it in the precedence list (topological ordering). The in-degree value of each of v's children is decremented by one. Once an operator has an in-degree value of zero in the array the operator is placed at the end of the queue of those operators waiting to be processed. As each operator is processed it is removed from the queue and placed in the precedence list. This process continues until the queue is empty. The new topological sort can handle input in any order.

3. FILE_PROCESSOR

This package, which processed the initial input and tested the input to determine if a the operators could be scheduled on a single processor system, now only tests the input and calculates periods for the non-periodic operators. This package is renamed PROCESSOR. Processing of the input now occurs in the packages FRONT_END and NEW_DATA_STRUCTURES.

4. Files

The original FILES package contained the definitions of all the types, instantiation of all the generic packages, and global variables used by the static scheduler. The new package contains the same type of information. This new package is named DATA. Since the data structures used by the static scheduler are different, the new package reflects these changes.

C PACKAGES REMOVED FROM THE STATIC SCHEDULER

During development of the new algorithm the existing data structures were examined. In addition to modifying several packages to improve their performance, several packages were eliminated because they were inefficient in their execution and thus were replaced by new packages. The removed packages are DIGRAPH, the HARMONIC_BLOCK_BUILDER scheduling algorithm, and OPERATOR_SCHEDULER.

The instantiation of the generic package GRAPHS resulted in the package DIGRAPH, which was a linked list representation of the operators and their precedence relationships. This package, once created, did not require any changes throughout the execution of the static scheduler. Using linked lists to represent graphs with their associated parent-child relationships is very inefficient. Numerous linked list traversals were required in order to determine the parents or children of a specific operator. The graph structure was not internal to this package but was passed as a parameter from procedure to procedure within the static scheduler increasing the input/output requirements. Procedures also existed within this package allowing for the removal and addition of nodes and edges in the graph. This could result in the unintentional removal or addition of information or changes to the relationships between operators. The generic package GRAPHS has been replaced by a new generic package NEW_DATA_STRUCTURES which is described in detail in the next section.

The HARMONIC_BLOCK_BUILDER scheduling algorithm is incorporated into the simulated annealing scheduling algorithm. The HARMONIC_BLOCK_BUILDER algorithm is used to develop the initial solution. If all the timing constraints are satisfied, simulated annealing does not occur since a legal schedule exists and the static scheduler terminates.

The OPERATOR_SCHEDULER package, which contained the routines TEST_DATA, the HARMONIC_BLOCK_BUILDER, EARLIEST_START, and EARLIEST_DEADLINE algorithms, is removed and replaced by the SCHEDULER package. The procedure TEST_DATA is moved to the package FRONT_END. Correct implementations of the EARLIEST_START and EARLIEST_DEADLINE scheduling algorithms that make use of the new packages and data structures are contained in the package SCHEDULER.

D. NEW PACKAGES AND DATA STRUCTURES

Several new packages and data structures are contained in the new version of the static scheduler. These modifications improve performance and correctness, streamline input/output, and simplify the static scheduler. These new packages are FRONT_END, NEW_DATA_STRUCTURES, PRIORITY QUEUE, SCHEDULER, and ANNEAL.

1. FRONT_END

This package contains the procedures PRODUCE_OP_LIST and TEST_DATA. The procedure PRODUCE_OP_LIST reads the text input file ATOMIC.INFO. Depending on the keywords, which are declared as constants, the procedure separates the information in the file and stores the time critical operator information in a linked list that is used by the package

NEW_DATA_STRUCTURES. This procedure also produce a count of the number of operators to be scheduled.

The procedure TEST_DATA, described in detail and implemented in (Janson, 1988) is also contained in this package. This allows the input to be examined as soon as a linked list of operators is established so that system resources are not wasted if a feasible schedule is not possible for a given input.

The new representation of the graph NEW_GRAPH, is instantiated from the generic package NEW_DATA_STRUCTURES, in the package FRONT_END. This allows for visibility of NEW_GRAPH by the rest of the packages within the static scheduler.

2. NEW_DATA_STRUCTURES

This generic package replaces the generic package GRAPHS. It represents an acyclic graph structure of operators of the hard real-time system in a simpler and easily accessible data structure. The new graph is a record that consists of two entries, OP_ARRAY and OP_MATRIX (see Figure 11). Unlike the old graphical representation all information about the operators; i.e their name, period, maximum execution time, etc. as well as their parent-child relationships only exist within this package. All relevant information about the operators that is required by the static scheduler is accessible by way of procedures and functions that are instantiated within the package and visible outside of it.

Since the operator information does not change once the new graph is created, the decision was made to streamline this data structure. Using the Ada principle of information hiding, the graph structure and its contents are private so that this information cannot inadvertently be changed. This was not the case with the old graph structure.

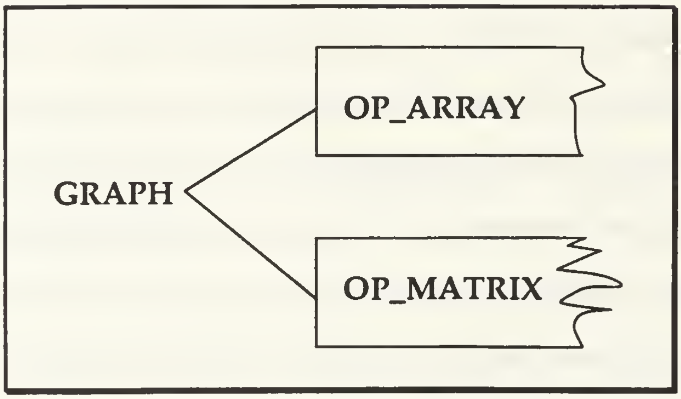


Figure 11. Graph Structure

The generic package NEW_DATA_STRUCTURES is instantiated in the declarative part of the package FRONT_END. However, OP_ARRAY and OP_MATRIX cannot be instantiated at this point because the number of operators to be scheduled is not known until ATOMIC.INFO is processed. By once again using the principles of Ada this is possible by creating the record structure called GRAPH and declaring a pointer type to this data structure. Once the number of operators to be scheduled is known the Ada allocator "new" is used to create an instance of GRAPH that contains the proper size OP_ARRAY and OP_MATRIX. This allows for efficient use of memory.

The data structure OP_ARRAY contains all relevant information about the operators. Once the operators are stored in the array they are identified by their index position in the array, which are integers. This allows for immediate access of all relevant operator information instead of having to traverse a linked list in order to find the desired operator. Identifying operators by their index position as opposed to their name reduces the storage required for operator identification throughout the static scheduler.

The data structure OP_MATRIX, which is a two dimensional array, greatly speeds up execution of the static scheduler. In the old graph data structure numerous linked lists traversals were required in order to determine the parent-child relationships of operators. The new graph data structure, illustrated in Figure 13, streamlines the execution of this requirement. Each operator has a row and column in the matrix. Each cell in the matrix has two entries, one for a parent operator and one for a child operator. The diagonal cells [i,i] in the matrix act as header nodes for two circularly linked lists, one containing the parents of node i in the graph, and the other containing the children of node i. For all i/=j, the child operator (respectively parent operator) field of the $[i,j]^{th}$ entry is -1 if OP_j is not a child (respectively parent) of OP_i. Otherwise, the child operator (respectively parent operator) field will contain the index number of the next child (respectively parent) in the circular linked list. For example, using Figures 13 anad 14, the children of Op_2 can be retrieved as follows: starting at cell [2,2] retrieve the value 5 from the corresponding child operator field. Moving to cell [2,5], retrieve the value 6 from the child position. Moving to cell [2,6] we see that there is a value of 2 in the child position, returning us back to the starting cell. At this point we know OP_2 has no more children. A similar routine is used to identify an operators parent's, only moves are made column wise as opposed to row wise. To check a parent-child relationship we can go right to the cell in question. If the value of the appropriate field is not -1, then a relationship exists.

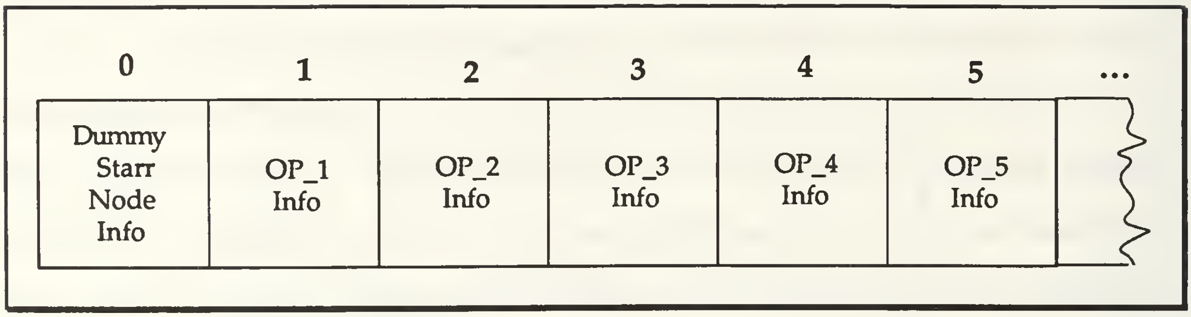
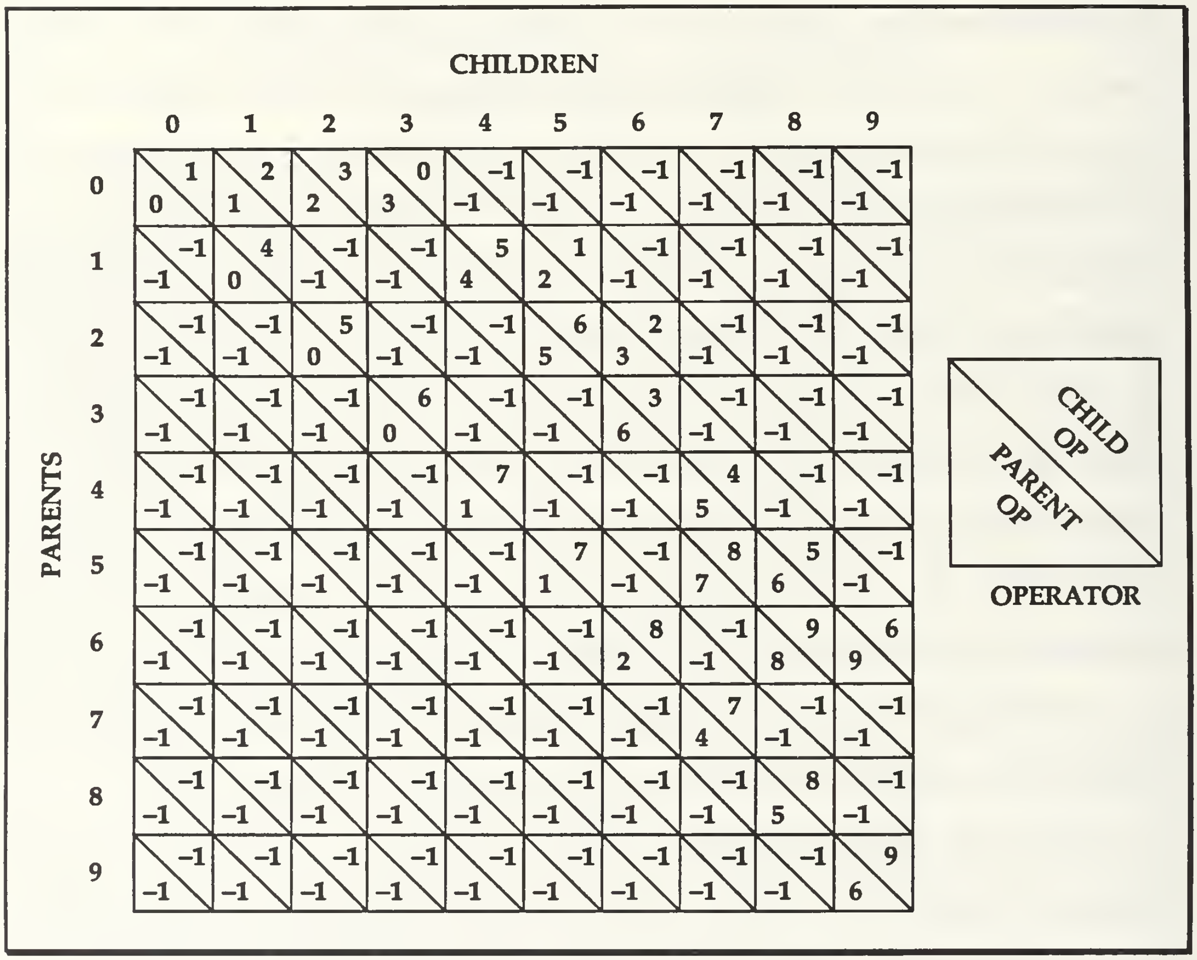


Figure 12. Operator Array



	0	1	2	3	4	5	6	7	8	9
0	1 0	2 1	3 2	0 3	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1
1	–1 –1	4 0	–1 –1	–1 –1	5 4	1 2	–1 –1	–1 –1	–1 –1	–1 –1
2	–1 –1	–1 –1	5 0	–1 –1	–1 –1	6 5	2 3	–1 –1	–1 –1	–1 –1
3	–1 –1	–1 –1	–1 –1	6 0	–1 –1	–1 –1	3 6	–1 –1	–1 –1	–1 –1
4	–1 –1	–1 –1	–1 –1	–1 –1	7 1	–1 –1	–1 –1	4 5	–1 –1	–1 –1
5	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	7 1	–1 –1	8 7	5 6	–1 –1
6	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	8 2	–1 –1	9 8	6 9
7	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	7 4	–1 –1	–1 –1
8	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	8 5	–1 –1
9	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	–1 –1	9 6

Figure 13. Operator Matrix

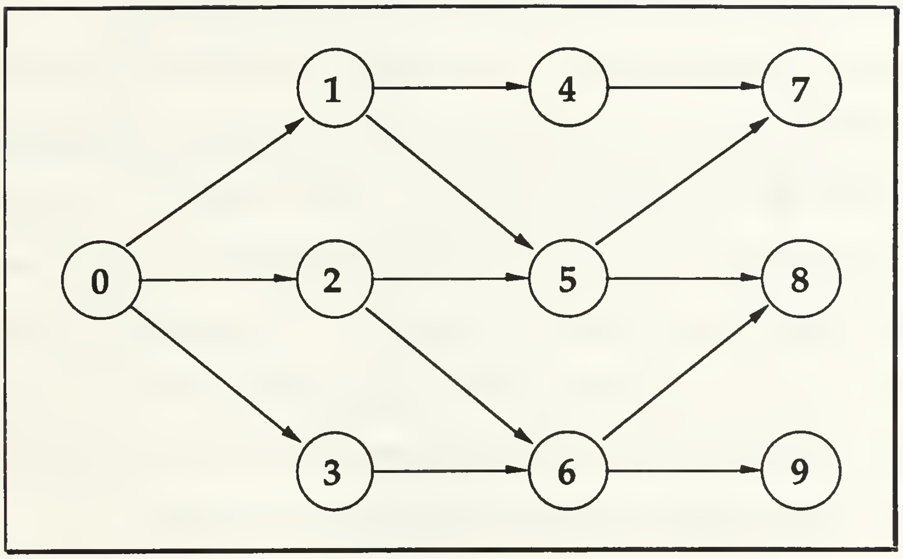


Figure 14. Matrix Representation of Graph

3. PRIORITY_QUEUE

This generic package is used by the earliest start and earliest deadline scheduling algorithms. During instantiation of this package three parameters are passed in to the generic template. The first is the type of element that is to be placed in the priority queue. The second is the type of the value used to order this priority queue. The third is the function used to order the priority queue. By using a priority queue the code for both the earliest start and earliest deadline algorithms is simplified. Under the Ada principle of code reusability, the generic priority queue package is a reusable software component that has a wide range of uses.

4. Anneal

This package contains the code for the new scheduling algorithm that is described in detail in Chapter III of this thesis. It contains all the necessary procedures and functions required to perform simulated annealing.

E. DESCRIPTION OF THE NEW STATIC SCHEDULER

The new implementation of the static scheduler still takes the same input, ATOMIC.INFO and produces the same output, the Ada textfile SS.a. Figure 15 shows the dataflow of the new static scheduler. As illustrated in Figure 15, once the input is stored in the new data structure, the requirement for cumbersome input/output is removed. All necessary information is accessible through the package NEW_DATA_STRUCTURE. The new static scheduler accomplishes the same functions as the old static scheduler, but it does so in a more efficient, simplified, and correct manner.

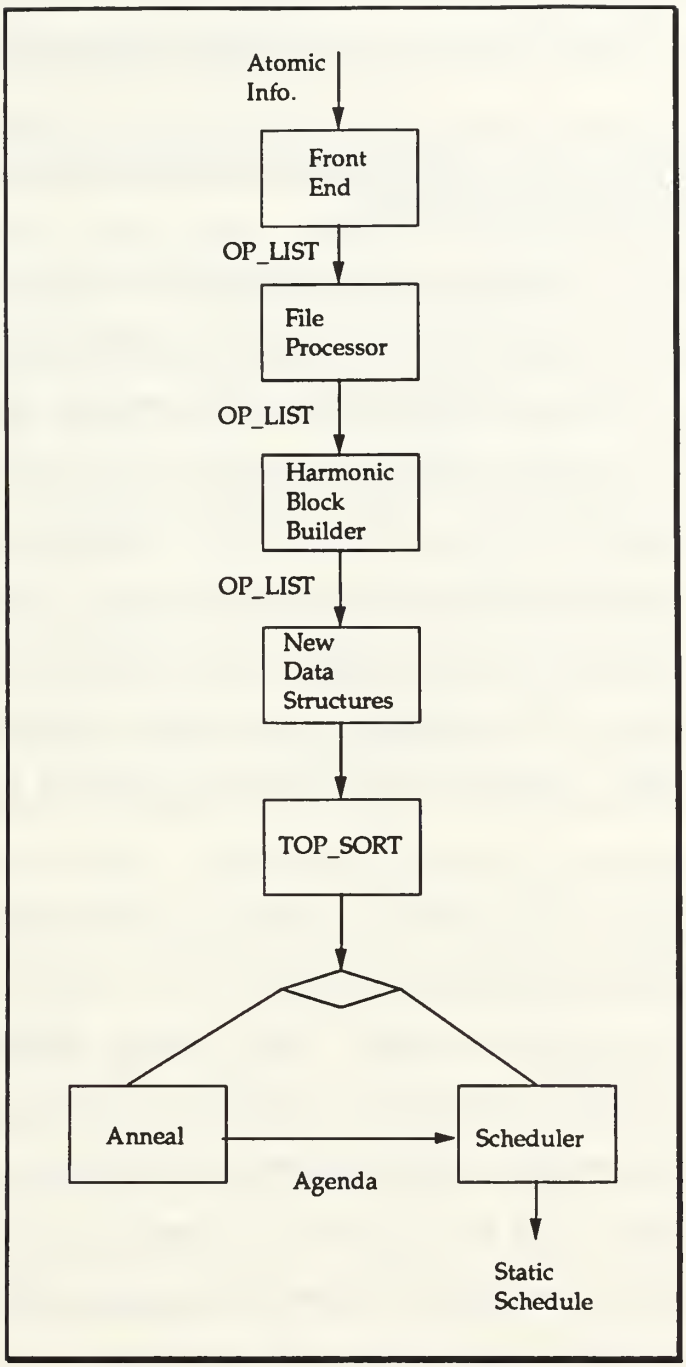


Figure 15. Data Flow Diagram of Static Schedule

V. EVALUATION OF THE NEW ALGORITHM

A. IMPROVEMENTS IN PERFORMANCE OF THE NEW ALGORITHM OVER PREVIOUS ALGORITHMS

The simulated annealing scheduling algorithm starts with an initial solution that satisfies the precedence constraints of the hard real-time system, and attempts to find a feasible solution that satisfies the system timing constraints. The simulated annealing algorithm is not intended to run faster than either the earliest start or earliest deadline scheduling algorithm. It is intended to find feasible schedules that cannot be found by the earliest start or earliest deadline algorithms and to serve as an alternative to the more costly branch and bound and exhaustive enumeration scheduling algorithms. Based on the initial testing results simulated annealing accomplishes this goal.

The performance and results of both the earliest start and earliest deadline scheduling algorithm improved as a result of the changes implemented in the static scheduler. These changes, discussed in detail in Chapter IV of this thesis, resulted in a rewriting of both of these algorithms. These algorithms now find correct solutions for cases that were not solved in the previous version of the static scheduler. In particular, the algorithm does not output incorrect schedules that exceed the harmonic block length, which they did in the previous version of the static scheduler.

Any new scheduling algorithm that is implemented in the static scheduler should utilize the packages and the data structures that were implemented as a result of this thesis. These data structures are efficient and do not require large amounts of memory.

B. EXAMINATION OF THE SIMULATED ANNEALING ALGORITHM ON HARD REAL-TIME SYSTEM PROBLEMS

The algorithm's initial performance in handling hard real-time system problems is satisfactory. Two test cases are presented in this thesis and the simulated annealing algorithm was able to find a feasible solution when both earliest start and earliest deadline scheduling algorithms failed to find a feasible solution. Simulated annealing was not costly time wise when it came to finding these solutions. This indicates that the parameters chosen for the simulated annealing scheduling algorithm (i.e. freezing temperature, cooling factor, the number of trials at each temperature) are satisfactory choices.

The first case consisted of eight operators. The input file and the precedence graph are included in Appendix A of this thesis and the results are presented in Table 1 below. This case was relatively simple in that there was a single starting node and there was not a wide variance in periods between the various operators. Due to the tight timing constraints both earliest start and earliest deadline were unable to find a solution. Simulated annealing, on the other hand, quickly found a feasible solution. By starting with an initial solution that did not satisfy the hard real-time systems timing constraints, simulated annealing adjusted the operators while maintaining operator precedence and quickly found a feasible solution. The solution satisfied all timing constraints, including the one failed by earliest start and earliest

TABLE 1. RESULTS OF THE FIRST TEST CASE

EARLIEST START

OPERATOR	START TIME	STOP TIME	LOWER	UPPER	
DUMMY START NODE	0	0	30010	0	
OP_1	0	2000	0	0	
OP_4	2000	3000	0	0	
OP_3	3000	8000	0	0	
OP_7	8000	9000	0	0	
OP_2	9000	10000	0	0	
OP_5	10000	13000	0	0	
OP_1	13000	15000	10000	0	
OP_6	15000	16000	0	0	
OP_8	16000	17000	0	0	
OP_1	20000	22000	20000	0	
OP_2	24000	25000	24000	0	
OP_5	25000	28000	25000	0	
OP_6	30000	31000	30000	0	← Violate Harmonic
OP_8	31000	32000	31000	0	← Block Length

EARLIEST DEADLINE

OPERATOR	START TIME	STOP TIME	LOWER	UPPER	
DUMMY START NODE	0	0	30010	0	
OP_1	0	2000	0	0	
OP_4	2000	3000	0	0	
OP_3	3000	8000	0	0	
OP_7	8000	9000	0	0	
OP_2	9000	10000	0	0	
OP_5	10000	13000	0	0	
OP_1	13000	15000	10000	17000	
OP_6	15000	16000	0	0	
OP_8	16000	17000	0	0	
OP_1	20000	22000	20000	27000	
OP_2	24000	25000	24000	33000	
OP_5	25000	28000	25000	33000	
OP_8	31000	32000	31000	40000	← Violate Harmonic
OP_6	32000	33000	30000	41000	← Block Length

SIMULATED ANNEALING

OPERATOR	START TIME	STOP TIME	LOWER	UPPER
DUMMY START NODE	0	0	30010	0
OP_1	0	2000	0	7000
OP_4	2000	3000	2000	16000
OP_3	3000	8000	3000	13000
OP_7	8000	9000	8000	25000
OP_2	9000	10000	9000	18000
OP_5	10000	13000	10000	18000
OP_6	13000	14000	13000	24000
OP_8	14000	15000	14000	23000
OP_1	15000	17000	10000	17000
OP_1	20000	22000	20000	27000
OP_2	24000	25000	24000	33000
OP_5	25000	28000	25000	33000
OP_6	28000	29000	28000	39000
OP_8	29000	30000	29000	38000

deadline, because they exceed the harmonic block length. The previous version of the static scheduler would have output these schedules as correct schedules, even though they are not correct.

The second test case is based on the functional specifications of the C^3I work station described in (Anderson, 1990) and implemented in Coskun, 1990). The input file and precedence graph are presented in Appendix B of this thesis and the results are presented in Table 2. This case is more complicated than the first test case. It consists of 19 time critical operators. There is no specific starting operator. Any one of five operators may begin execution at the start of the harmonic block. There is a variance in periods between the various operators. The precedence relationships in this example are more complicated than the first case. As in the first case, due to the tight timing constraints, earliest start and earliest deadline fail to find a feasible schedule. Simulated annealing, however, is able to rapidly find a feasible schedule.

These two test cases indicate that simulated annealing shows promising results in solving the hard real-time scheduling problem. It appears that simulated annealing will perform well as a scheduling tool when both earliest start and earliest deadline fail. The cost of using simulated annealing is low enough for it to be used before trying a more costly enumeration algorithm.

TABLE 2. RESULTS OF THE SECOND TEST CASE

	EARLIEST START			
OPERATOR	START TIME	STOP TIME	LOWER	UPPER
DUMMY START NODE	0	0	21010	0
WEAPONS_SYSTEMS	0	100	0	0
WEAPONS_INTERFACE	100	200	0	0
CREATE_POSITION_DATA	200	700	0	0
MONITOR_OWNSHIP_POSITION	700	1200	0	0
CREATE_SENSOR_DATA	1200	1300	0	0
ANALYZE_SENSOR_DATA	1300	1550	0	0
PREPARE_SENSOR_TRACK	1550	1800	0	0
FILTER_SENSOR_TRACKS	1800	2300	0	0
ADD_SENSOR_TRACK	2300	2800	0	0
PREPARE_PERIODIC_REPORT	2800	3300	0	0
WEAPONS_SYSTEMS	3300	3400	3000	0
WEAPONS_INTERFACE	3400	3500	3100	0
CREATE_POSITION_DATA	3500	4000	3200	0
MONITOR_OWNSHIP_POSITION	4000	4500	3700	0
MAKE_ROUTING	4500	4800	0	0
FORWARD_FOR_TRANSMISSION	4800	4900	0	0
CONVERT_TO_TEXT_FILE	4900	5000	0	0
COMMS_LINKS	5000	5100	0	0
PARSE_INPUT_FILE	5100	5350	0	0
DECIDE_FOR_ARCHIVING	5350	5450	0	0
EXTRACT_TRACKS	5450	5600	0	0
FILTER_COMMS_TRACKS	5600	6100	0	0
WEAPONS_SYSTEMS	6100	6200	6000	0
WEAPONS_INTERFACE	6200	6300	6100	0
CREATE_POSITION_DATA	6300	6800	6200	0
MONITOR_OWNSHIP_POSITION	6800	7300	6700	0
ADD_COMMS_TRACK	7300	7400	0	0
CREATE_SENSOR_DATA	8200	8300	8200	0
ANALYZE_SENSOR_DATA	8300	8550	8300	0
PREPARE_SENSOR_TRACK	8550	8800	8550	0
FILTER_SENSOR_TRACKS	8800	9300	8800	0
WEAPONS_SYSTEMS	9300	9400	9000	0
WEAPONS_INTERFACE	9400	9500	9100	0
CREATE_POSITION_DATA	9500	10000	9200	0
ADD_SENSOR_TRACK	10000	10500	9300	0
MONITOR_OWNSHIP_POSITION	10500	11000	9700	0
PREPARE_PERIODIC_REPORT	11000	11500	9800	0
MAKE_ROUTING	11500	11800	11500	0
FORWARD_FOR_TRANSMISSION	11800	11900	11800	0
CONVERT_TO_TEXT_FILE	11900	12000	11900	0
COMMS_LINKS	12000	12100	12000	0
WEAPONS_SYSTEMS	12100	12200	12000	0
PARSE_INPUT_FILE	12200	12450	12100	0
WEAPONS_INTERFACE	12450	12550	12100	0
CREATE_POSITION_DATA	12550	13050	12200	0
DECIDE_FOR_ARCHIVING	13050	13150	12350	0
EXTRACT_TRACKS	13150	13300	12450	0
FILTER_COMMS_TRACKS	13300	13800	12600	0
MONITOR_OWNSHIP_POSITION	13800	14300	12700	0
ADD_COMMS_TRACK	14300	14400	14300	0
WEAPONS_SYSTEMS	15000	15100	15000	0
WEAPONS_INTERFACE	15100	15200	15100	0
CREATE_SENSOR_DATA	15200	15300	15200	0
CREATE_POSITION_DATA	15300	15800	15200	0
ANALYZE_SENSOR_DATA	15800	16050	15300	0
PREPARE_SENSOR_TRACK	16050	16300	15550	0

TABLE 2. RESULTS OF THE SECOND TEST CASE (CONTINUED)

```
MONITOR_OWNSHIP_POSITION        16300      16800      15700          0
FILTER_SENSOR_TRACKS            16800      17300      15800          0
ADD_SENSOR_TRACK                17300      17800      16300          0
PREPARE_PERIODIC_REPORT         17800      18300      16800          0
WEAPONS_SYSTEMS                 18300      18400      18000          0
WEAPONS_INTERFACE               18400      18500      18100          0
CREATE_POSITION_DATA            18500      19000      18200          0
MAKE_ROUTING                    19000      19300      18500          0
MONITOR_OWNSHIP_POSITION        19300      19800      18700          0
FORWARD_FOR_TRANSMISSION        19800      19900      18800          0
CONVERT_TO_TEXT_FILE            19900      20000      18900          0
COMMS_LINKS                     20000      20100      19000          0
PARSE_INPUT_FILE                20100      20350      19100          0
DECIDE_FOR_ARCHIVING            20350      20450      19350          0
EXTRACT_TRACKS                  20450      20600      19450          0
FILTER_COMMS_TRACKS             20600      21100      19600          0  ← Violate Harmonic
                                                                            Block Length

                         EARLIEST DEADLINE
THE BEST SCHEDULE FOLLOWS:
OPERATOR                   START TIME STOP TIME      LOWER      UPPER
DUMMY START NODE                    0         0      21010          0
WEAPONS_SYSTEMS                     0       100          0          0
WEAPONS_INTERFACE                 100       200          0          0
CREATE_POSITION_DATA              200       700          0          0
MONITOR_OWNSHIP_POSITION          700      1200          0          0
CREATE_SENSOR_DATA               1200      1300          0          0
ANALYZE_SENSOR_DATA              1300      1550          0          0
PREPARE_SENSOR_TRACK             1550      1800          0          0
FILTER_SENSOR_TRACKS             1800      2300          0          0
ADD_SENSOR_TRACK                 2300      2800          0          0
PREPARE_PERIODIC_REPORT          2800      3300          0          0
CREATE_POSITION_DATA             3300      3800       3200       5700
WEAPONS_SYSTEMS                  3800      3900       3000       5900
WEAPONS_INTERFACE                3900      4000       3100       6000
MONITOR_OWNSHIP_POSITION         4000      4500       3700       6200
MAKE_ROUTING                     4500      4800          0          0
FORWARD_FOR_TRANSMISSION         4800      4900          0          0
CONVERT_TO_TEXT_FILE             4900      5000          0          0
COMMS_LINKS                      5000      5100          0          0
PARSE_INPUT_FILE                 5100      5350          0          0
DECIDE_FOR_ARCHIVING             5350      5450          0          0
EXTRACT_TRACKS                   5450      5600          0          0
FILTER_COMMS_TRACKS              5600      6100          0          0
ADD_COMMS_TRACK                  6100      6200          0          0
CREATE_POSITION_DATA             6200      6700       6200       8700
WEAPONS_SYSTEMS                  6700      6800       6000       8900
WEAPONS_INTERFACE                6800      6900       6100       9000
MONITOR_OWNSHIP_POSITION         6900      7400       6700       9200
CREATE_POSITION_DATA             9200      9700       9200      11700
WEAPONS_SYSTEMS                  9700      9800       9000      11900
WEAPONS_INTERFACE                9800      9900       9100      12000
MONITOR_OWNSHIP_POSITION         9900     10400       9700      12200
CREATE_POSITION_DATA            12200     12700      12200      14700
WEAPONS_SYSTEMS                 12700     12800      12000      14900
WEAPONS_INTERFACE               12800     12900      12100      15000
ANALYZE_SENSOR_DATA             12900     13150       8300      15050
CREATE_SENSOR_DATA              13150     13250       8200      15100
```

TABLE 2. RESULTS OF THE SECOND TEST CASE (CONTINUED)

MONITOR_OWNSHIP_POSITION	13250	13750	12700	15200	
PREPARE_SENSOR_TRACK	13750	14000	8550	15300	
FILTER_SENSOR_TRACKS	14000	14500	8800	15300	
ADD_SENSOR_TRACK	14500	15000	9300	15800	
PREPARE_PERIODIC_REPORT	15000	15500	9800	16300	
CREATE_POSITION_DATA	15500	16000	15200	17700	
WEAPONS_SYSTEMS	16000	16100	15000	17900	
WEAPONS_INTERFACE	16100	16200	15100	18000	
MAKE_ROUTING	16200	16500	11500	18200	
MONITOR_OWNSHIP_POSITION	16500	17000	15700	18200	
FORWARD_FOR_TRANSMISSION	17000	17100	11800	18700	
CONVERT_TO_TEXT_FILE	17100	17200	11900	18800	
PARSE_INPUT_FILE	17200	17450	12100	18850	
COMMS_LINKS	17450	17550	12000	18900	
FILTER_COMMS_TRACKS	17550	18050	12600	19100	
DECIDE_FOR_ARCHIVING	18050	18150	12350	19250	
EXTRACT_TRACKS	18150	18300	12450	19300	
ADD_COMMS_TRACK	18300	18400	13100	20000	
CREATE_POSITION_DATA	18400	18900	18200	20700	
WEAPONS_SYSTEMS	18900	19000	18000	20900	
WEAPONS_INTERFACE	19000	19100	18100	21000	
MONITOR_OWNSHIP_POSITION	19100	19600	18700	21200	
ANALYZE_SENSOR_DATA	19600	19850	15300	22050	
CREATE_SENSOR_DATA	19850	19950	15200	22100	
PREPARE_SENSOR_TRACK	19950	20200	15550	22300	
FILTER_SENSOR_TRACKS	20200	20700	15800	22300	
ADD_SENSOR_TRACK	20700	21200	16300	22800	←
PREPARE_PERIODIC_REPORT	21200	21700	16800	23300	←
MAKE_ROUTING	21700	22000	18500	25200	←
FORWARD_FOR_TRANSMISSION	22000	22100	18800	25700	←
CONVERT_TO_TEXT_FILE	22100	22200	18900	25800	←
PARSE_INPUT_FILE	22200	22450	19100	25850	←
COMMS_LINKS	22450	22550	19000	25900	←
FILTER_COMMS_TRACKS	22550	23050	19600	26100	←
DECIDE_FOR_ARCHIVING	23050	23150	19350	26250	←
EXTRACT_TRACKS	23150	23300	19450	26300	←
ADD_COMMS_TRACK	23300	23400	20100	27000	←

Violate
Harmonic
Block
Length

Simulated Annealing

OPERATOR	START TIME	STOP TIME	LOWER	UPPER
DUMMY START NODE	0	0	21010	0
CREATE_POSITION_DATA	0	500	0	2500
CREATE_SENSOR_DATA	500	600	500	7400
WEAPONS_SYSTEMS	600	700	600	3500
ANALYZE_SENSOR_DATA	700	950	700	7450
COMMS_LINKS	950	1050	950	7850
WEAPONS_INTERFACE	1050	1150	1050	3950
MONITOR_OWNSHIP_POSITION	1150	1650	1150	3650
PREPARE_SENSOR_TRACK	1650	1900	1650	8400
FILTER_SENSOR_TRACKS	1900	2400	1900	8400
ADD_SENSOR_TRACK	2400	2900	2400	8900
PREPARE_PERIODIC_REPORT	2900	3400	2900	9400
MAKE_ROUTING	3400	3700	3400	10100
WEAPONS_SYSTEMS	3700	3800	3600	6500
FORWARD_FOR_TRANSMISSION	3800	3900	3700	10600
CONVERT_TO_TEXT_FILE	3900	4000	3800	10700
PARSE_INPUT_FILE	4000	4250	3900	10650

TABLE 2. RESULTS OF THE SECOND TEST CASE (CONTINUED)

WEAPONS_INTERFACE	4250	4350	4050	6950
DECIDE_FOR_ARCHIVING	4350	4450	4150	11050
EXTRACT_TRACKS	4450	4600	4250	11100
FILTER_COMMS_TRACKS	4600	5100	4400	10900
ADD_COMMS_TRACK	5100	5200	4900	11800
CREATE_POSITION_DATA	5200	5700	3000	5500
MONITOR_OWNSHIP_POSITION	5700	6200	4150	6650
CREATE_POSITION_DATA	6200	6700	6000	8500
WEAPONS_SYSTEMS	6700	6800	6600	9500
WEAPONS_INTERFACE	7050	7150	7050	9950
MONITOR_OWNSHIP_POSITION	7150	7650	7150	9650
CREATE_SENSOR_DATA	7650	7750	7500	14400
ANALYZE_SENSOR_DATA	7750	8000	7700	14450
COMMS_LINKS	8000	8100	7950	14850
PREPARE_SENSOR_TRACK	8650	8900	8650	15400
FILTER_SENSOR_TRACKS	8900	9400	8900	15400
CREATE_POSITION_DATA	9400	9900	9000	11500
WEAPONS_SYSTEMS	9900	10000	9600	12500
ADD_SENSOR_TRACK	10000	10500	9400	15900
MONITOR_OWNSHIP_POSITION	10500	11000	10150	12650
WEAPONS_INTERFACE	11000	11100	10050	12950
PREPARE_PERIODIC_REPORT	11100	11600	9900	16400
MAKE_ROUTING	11600	11900	10400	17100
FORWARD_FOR_TRANSMISSION	11900	12000	10700	17600
CONVERT_TO_TEXT_FILE	12000	12100	10800	17700
CREATE_POSITION_DATA	12100	12600	12000	14500
PARSE_INPUT_FILE	12600	12850	10900	17650
WEAPONS_SYSTEMS	12850	12950	12600	15500
DECIDE_FOR_ARCHIVING	12950	13050	11150	18050
EXTRACT_TRACKS	13050	13200	11250	18100
MONITOR_OWNSHIP_POSITION	13200	13700	13150	15650
WEAPONS_INTERFACE	13700	13800	13050	15950
FILTER_COMMS_TRACKS	13800	14300	11400	17900
ADD_COMMS_TRACK	14300	14400	11900	18800
CREATE_SENSOR_DATA	14500	14600	14500	21400
ANALYZE_SENSOR_DATA	14700	14950	14700	21450
COMMS_LINKS	14950	15050	14950	21850
CREATE_POSITION_DATA	15050	15550	15000	17500
WEAPONS_SYSTEMS	15600	15700	15600	18500
PREPARE_SENSOR_TRACK	15700	15950	15650	22400
FILTER_SENSOR_TRACKS	15950	16450	15900	22400
MONITOR_OWNSHIP_POSITION	16450	16950	16150	18650
WEAPONS_INTERFACE	16950	17050	16050	18950
ADD_SENSOR_TRACK	17050	17550	16400	22900
PREPARE_PERIODIC_REPORT	17550	18050	16900	23400
CREATE_POSITION_DATA	18050	18550	18000	20500
MAKE_ROUTING	18550	18850	17400	24100
WEAPONS_SYSTEMS	18850	18950	18600	21500
FORWARD_FOR_TRANSMISSION	18950	19050	17700	24600
CONVERT_TO_TEXT_FILE	19050	19150	17800	24700
WEAPONS_INTERFACE	19150	19250	19050	21950
MONITOR_OWNSHIP_POSITION	19250	19750	19150	21650
PARSE_INPUT_FILE	19750	20000	17900	24650
DECIDE_FOR_ARCHIVING	20000	20100	18150	25050
EXTRACT_TRACKS	20100	20250	18250	25100
FILTER_COMMS_TRACKS	20250	20750	18400	24900
ADD_COMMS_TRACK	20750	20850	18900	25800

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis intended to develop a fast heuristic static scheduling algorithm. Simulated annealing was chosen as a basis for developing such a static scheduling algorithm because of the promising results simulated annealing demonstrated in solving other NP-Hard type problems. Simulated annealing proved to be useful in solving optimization type problems, and the development of hard real-time schedules is a subclass of this type of problem. The initial results of the simulated annealing static scheduling algorithm are promising.

The major emphasis of this thesis was the development of a new static scheduling algorithm. In addition, this thesis built on previous research conducted during the development of the static scheduler. Modifications made to data structures and scheduling algorithms already implemented improved the performance of the static scheduler portion of CAPS. Several of the new packages and data structures are generic in nature and are available to be used beyond the scope of this thesis. This is possible due to the use of the Ada principles of modularity and software reusability.

This thesis provides a running static scheduler that offers several choices of algorithms to use to find a feasible static schedule. Additional algorithms can be added to the static scheduler by using the data structures developed for this thesis. Additional research and development can continue to build on the work done in this thesis.

B. RECOMMENDATIONS

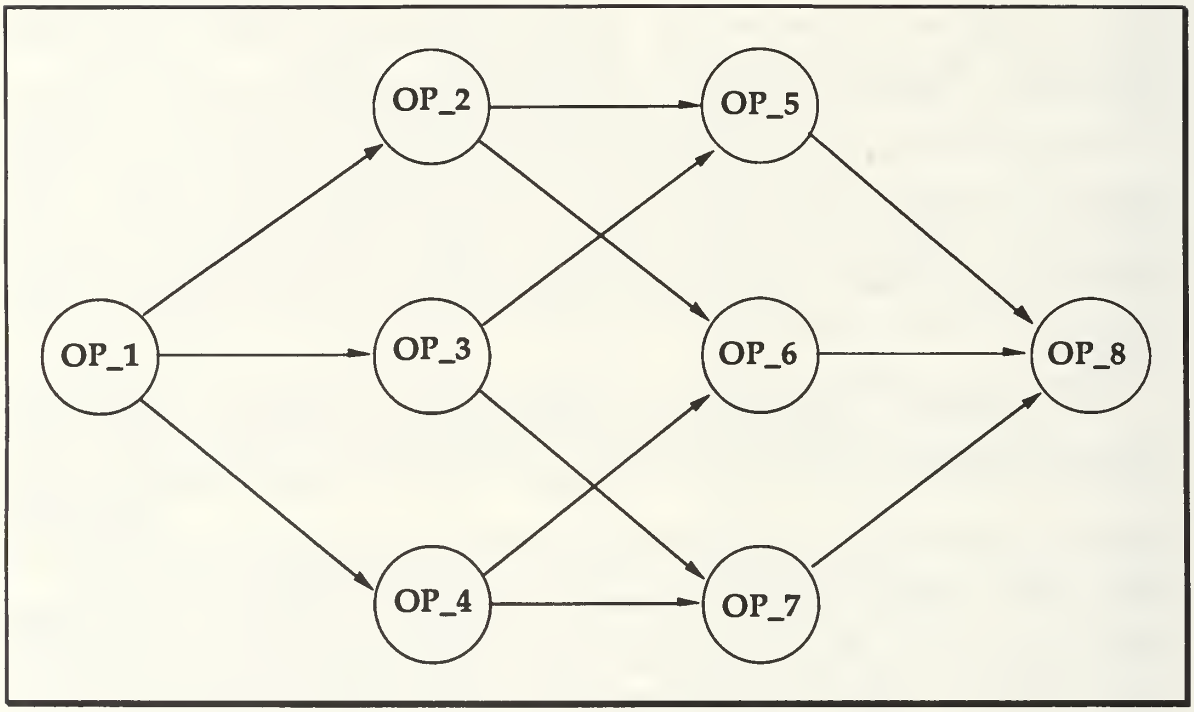
As a result of this thesis, several weaknesses and areas requiring improvement within the static scheduler were identified. Many shortcomings were corrected, but others require further effort. Due to the complexity of the static scheduler, all problems identified were not corrected.

In the current static scheduler, no differentiation is made between data flow and sampled stream data links. The performance and results of all scheduling algorithms would most likely improve if this information were utilized.

The algorithm described and implemented in (Coskun, 1990) that calculates periodic equivalents for non-periodic time critical operators merits further examination. This algorithm is based on a theorem described in (Mok, 1985). Linked list data structures are used in the algorithm when arrays could suffice, saving execution time. Four separate linked list traversals are made in this algorithm. The performance and output of this indicate that it could be improved.

The development of a PSDL data type implemented in Ada will simplify the package FRONT_END described in Chapter IV of this thesis. When this package is available (see S. Baromoglu, *The Design and Implementation of an Expander for the Hierarchical Real-Time Constraints of Computer-Aided Prototyping System (CAPS)*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1991) the FRONT_END package should be modified to use the PSDL datatypes to provide input to the static scheduler. Once this occurs, the requirement for the ATOMIC.INFO file becomes unnecessary.

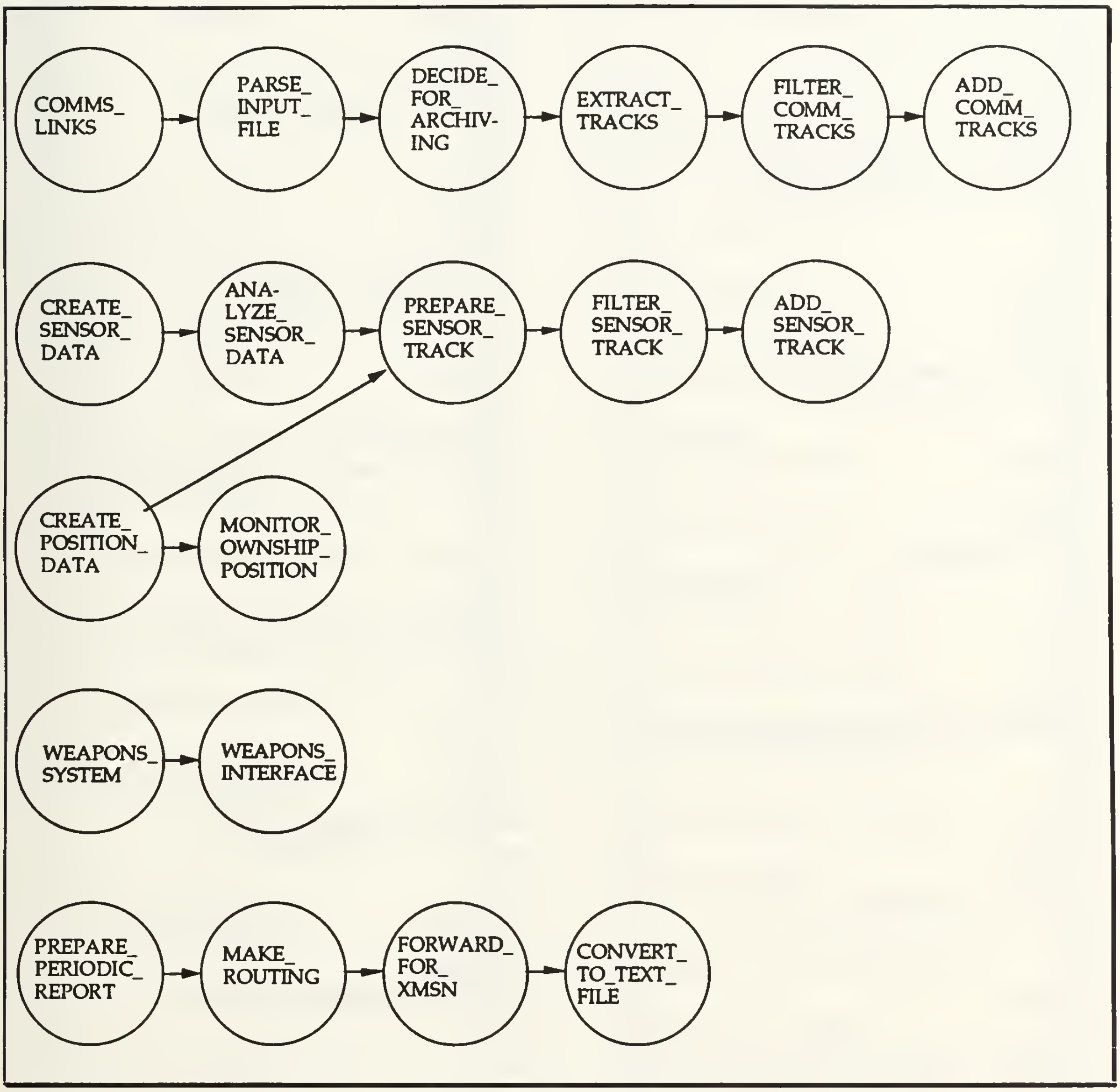
APPENDIX A. CASE 1 TEST DATA



Precedence Graph, Test Case 1

```
ATOMIC
OP_1
MET
2000
PERIOD
10000
WITHIN
9000
ATOMIC
OP_2
MET
1000
PERIOD
15000
WITHIN
10000
ATOMIC
OP_3
MET
5000
PERIOD
30000
WITHIN
15000
ATOMIC
OP_4
MET
1000
PERIOD
30000
WITHIN
15000
ATOMIC
OP_5
MET
3000
PERIOD
15000
WITHIN
11000
ATOMIC
OP_6
MET
1000
PERIOD
15000
WITHIN
12000
ATOMIC
OP_7
MET
1000
PERIOD
30000
WITHIN
18000
ATOMIC
OP_8
MET
1000
PERIOD
15000
WITHIN
10000
LINK
A
OP_1
0
OP_2
LINK
B
OP_1
0
OP_3
LINK
C
OP_1
0
OP_4
LINK
D
OP_2
0
OP_6
LINK
E
OP_2
0
OP_5
LINK
F
OP_3
0
OP_5
LINK
G
OP_3
0
OP_7
LINK
H
OP_4
0
OP_6
LINK
I
OP_4
0
OP_7
LINK
J
OP_5
0
OP_8
LINK
K
OP_7
0
OP_8
LINK
L
OP_6
0
OP_8
```

APPENDIX B. CASE 2 TEST DATA



Precedence graph, Case 2

```
ATOMIC
COMMS_LINKS
MET
100
PERIOD
7000
ATOMIC
PARSE_INPUT_FILE
MET
250
PERIOD
7000
ATOMIC
DECIDE_FOR_ARCHIVING
MET
100
PERIOD
7000
ATOMIC
EXTRACT_TRACKS
MET
150
PERIOD
7000
ATOMIC
MAKE_ROUTING
MET
300
PERIOD
7000
ATOMIC
FORWARD_FOR_TRANSMISSION
MET
100
PERIOD
7000
ATOMIC
CONVERT_TO_TEXT_FILE
MET
100
PERIOD
7000
ATOMIC
PREPARE_PERIODIC_REPORT
MET
500
PERIOD
7000
ATOMIC
FILTER_COMMS_TRACKS
MET
500
PERIOD
7000
ATOMIC
ADD_COMMS_TRACK
MET
100
PERIOD
7000
ATOMIC
FILTER_SENSOR_TRACKS
MET
500
PERIOD
7000
ATOMIC
ADD_SENSOR_TRACK
MET
500
PERIOD
7000
ATOMIC
MONITOR_OWNSHIP_POSITION
MET
500
PERIOD
3000
ATOMIC
CREATE_SENSOR_DATA
MET
100
PERIOD
7000
ATOMIC
ANALYZE_SENSOR_DATA
MET
250
PERIOD
7000
ATOMIC
PREPARE_SENSOR_TRACK
MET
250
PERIOD
7000
ATOMIC
CREATE_POSITION_DATA
MET
500
PERIOD
3000
```

```
ATOMIC
WEAPONS_INTERFACE
MET
100
PERIOD
3000
ATOMIC
WEAPONS_SYSTEMS
MET
100
PERIOD
3000
ATOMIC
DISPLAY_TRACKS
ATOMIC
GET_USER_INPUTS
ATOMIC
MANAGE_USER_INTERFACE
ATOMIC
STATUS_SCREEN
ATOMIC
EMERGENCY_STATUS_SCREEN
ATOMIC
MESSAGE_EDITOR
ATOMIC
MESSAGE_ARRIVAL_PANEL
LINK
INPUT_LINK_MESSAGE
COMMS_LINKS
1200
PARSE_INPUT_FILE
LINK
INPUT_TEXT_RECORD
PARSE_INPUT_FILE
500
DECIDE_FOR_ARCHIVING
LINK
TDD_ARCHIVE_SETUP
GET_USER_INPUTS
0
DECIDE_FOR_ARCHIVING
LINK
COMMS_TEXT_FILE
DECIDE_FOR_ARCHIVING
500
EXTRACT_TRACKS
LINK
COMMS_EMAIL
DECIDE_FOR_ARCHIVING
500
MESSAGE_ARRIVAL_PANEL
```

```
LINK
COMMS_ADD_TRACK
EXTRACT_TRACKS
500
FILTER_COMMS_TRACKS
LINK
TDD_FILTER
GET_USER_INPUTS
0
FILTER_COMMS_TRACKS
LINK
FILTERED_COMMS_TRACK
FILTER_COMMS_TRACKS
500
ADD_COMMS_TRACK
LINK
TDD_FILTER
GET_USER_INPUTS
0
ADD_COMMS_TRACK
LINK
OUT_TRACKS
ADD_COMMS_TRACK
500
DISPLAY_TRACKS
LINK
SENSOR_DATA
CREATE_SENSOR_DATA
800
ANALYZE_SENSOR_DATA
LINK
SENSOR_CONTACT_DATA
ANALYZE_SENSOR_DATA
500
PREPARE_SENSOR_TRACK
LINK
POSITION_DATA
CREATE_POSITION_DATA
800
PREPARE_SENSOR_TRACK
LINK
SENSOR_ADD_TRACK
PREPARE_SENSOR_TRACK
500
FILTER_SENSOR_TRACKS
LINK
TDD_FILTER
GET_USER_INPUTS
0
FILTER_SENSOR_TRACKS
LINK
```

```
FILTERED_SENSOR_TRACK
FILTER_SENSOR_TRACKS
500
ADD_SENSOR_TRACK
LINK
TDD_FILTER
GET_USER_INPUTS
0
ADD_SENSOR_TRACK
LINK
OUT_TRACKS
ADD_SENSOR_TRACK
500
DISPLAY_TRACKS
LINK
POSITION_DATA
CREATE_POSITION_DATA
800
MONITOR_OWNSHIP_POSITION
LINK
TD_TRACK_REQUEST
GET_USER_INPUTS
0
DISPLAY_TRACKS
LINK
OUT_TRACKS
MONITOR_OWNSHIP_POSITION
500
DISPLAY_TRACKS
LINK
WEAPON_STATUS_DATA
WEAPONS_SYSTEMS
500
WEAPONS_INTERFACE
LINK
WEAPONS_STATREP
WEAPONS_INTERFACE
500
STATUS_SCREEN
LINK
TCD_STATUS_QUERY
GET_USER_INPUTS
0
STATUS_SCREEN
LINK
WEAPONS_EMREP
WEAPONS_INTERFACE
500
EMERGENCY_STATUS_SCREEN
LINK
EDITOR_SELECTED
GET_USER_INPUTS
0
MESSAGE_EDITOR
LINK
TCD_TRANSMIT_COMMAND
MESSAGE_EDITOR
0
MAKE_ROUTING
LINK
TCD_NETWORK_SETUP
GET_USER_INPUTS
0
MAKE_ROUTING
LINK
TRANSMISSION_MESSAGE
MAKE_ROUTING
500
FORWARD_FOR_TRANSMISSION
LINK
TCD_EMISSION_CONTROL
GET_USER_INPUTS
0
FORWARD_FOR_TRANSMISSION
LINK
OUTPUT_MESSAGES
FORWARD_FOR_TRANSMISSION
500
CONVERT_TO_TEXT_FILE
LINK
INITIATE_TRANS
GET_USER_INPUTS
0
PREPARE_PERIODIC_REPORT
LINK
TERMINATE_TRANS
GET_USER_INPUTS
0
PREPARE_PERIODIC_REPORT
LINK
TCD_TRANSMIT_COMMAND
PREPARE_PERIODIC_REPORT
800
MAKE_ROUTING
```

APPENDIX C. MODIFIED PACKAGES

```
with VSTRINGS;
with SEQUENCES;
with TEXT_IO;

--* This package contains all of the global declarations and definitions
--* of data structures that are necessary for the Static Scheduler

package DATA is

  package VARSTRING is new VSTRINGS(80);
  use VARSTRING;
  subtype OPERATOR_ID is VSTRING;
  subtype VALUE is NATURAL;
  subtype MET is VALUE;
  subtype MRT is VALUE;
  subtype MCP is VALUE;
  subtype PERIOD is VALUE;
  subtype WITHIN is VALUE;
  subtype STARTS is VALUE;
  subtype STOPS is VALUE;
  subtype LOWERS is VALUE;
  subtype UPPERS is VALUE;

  Exception_Operator : OPERATOR_ID;

  TEST_VERIFIED : BOOLEAN := TRUE;

  type OPERATOR is
    record
      THE_OPERATOR_ID          : OPERATOR_ID;
      THE_MET                  : MET := 0;
      THE_MRT                  : MRT := 0;
      THE_MCP                  : MCP := 0;
      THE_PERIOD               : PERIOD := 0;
    THE_WITHIN                 : WITHIN := 0;
   end record;

  package V_LISTS is new SEQUENCES(OPERATOR);
  use V_LISTS;

  type SCHEDULE_INPUTS is
    record
      THE_OPERATOR             : INTEGER;
      THE_START                : STARTS := 0;
      THE_STOP                 : STOPS := 0;
      THE_LOWER                : LOWERS := 0;
      THE_UPPER                : UPPERS := 0;
      THE_INSTANCE             : INTEGER := 1;
  end record;

  package SCHEDULE_INPUTS_LIST is new SEQUENCES(SCHEDULE_INPUTS);

  package NODE_LIST is new SEQUENCES(INTEGER);

  NON_CRITS              : TEXT_IO.FILE_TYPE;
  AG_OUTFILE             : TEXT_IO.FILE_TYPE;
  INPUT                  : TEXT_IO.FILE_MODE := TEXT_IO.IN_FILE;
  OUTPUT                 : TEXT_IO.FILE_MODE := TEXT_IO.OUT_FILE;
```

```
   Current_Value       : VALUE;
   New_Word            : VARSTRING.VSTRING;
   Cur_Opt             : OPERATOR;

   OP_COUNT           : INTEGER;
   OP_LIST            : V_LISTS.LIST;

end DATA;
```

```
generic
  type ITEM is private;

  package SEQUENCES is

  type NODE;
  type LIST is access NODE;
  type NODE is
    record
      ELEMENT      : ITEM;
      NEXT         : LIST := null;
      PREVIOUS     : LIST := null; --* (APR 91)
    end record;

  BAD_VALUE : exception;

  function EQUAL(L1 : in LIST; L2 : in LIST) return BOOLEAN;

  procedure EMPTY(L : out LIST);

  function NON_EMPTY(L : in LIST) return BOOLEAN;

  function SUBSEQUENCE(L1 : in LIST; L2 : in LIST) return BOOLEAN;

  function MEMBER(X : in ITEM; L : in LIST) return BOOLEAN;

   procedure ADD(X : in ITEM; L : in out LIST);

   procedure REMOVE(X : in ITEM; L : in out LIST);

  procedure LIST_REVERSE(L1 : in LIST; L2 : in out LIST);

   procedure FREE_LIST(L: in out LIST);
  --* (JUL 91) Used by annealling and Exhaustive Enumeration to reclaim
  --* memory space that is no longer needed.

  procedure DUPLICATE(L1 : in LIST; L2 : in out LIST);

  function LOOK4(X : in ITEM; L : in LIST) return LIST;

  procedure NEXT(L : in out LIST);

  procedure PREVIOUS(L : in out LIST);
 --* (APR 91) Used by annealling

  function VALUE(L : in LIST) return ITEM;

  procedure INSERT_NEXT(X : in ITEM; L : in out LIST);
  --* (June 91) Item is inserted in proper position in list

  procedure REPLACE_ITEM(X : in ITEM; L : in out LIST);
  --* (JUL 91) Used by annealling

  procedure COPY_LIST(L1 : in LIST; L2 : in out LIST);
  --* (JUL 91) Used by annealling to reclaim memory that is no longer needed


end SEQUENCES;
```

```
with UNCHECKED_DEALLOCATION;
with TEXT_IO; --* test(Apr 91)

package body SEQUENCES is

   pragma LINK_WITH(“heaplib.sparc.ar”);

    procedure FREE is new UNCHECKED_DEALLOCATION(NODE, LIST);


   function NON_EMPTY(L : in LIST) return BOOLEAN is
   begin
      if L = null then
         return FALSE;
      else
         return TRUE;
      end if;
   end NON_EMPTY;

   procedure NEXT(L : in out LIST) is
   begin
      if L /= null then
         L := L.NEXT;
      end if;
 end NEXT;

    procedure PREVIOUS(L : in out LIST) is --* This procedure was added 10 Apr 91
    begin                                  --* to allow the annealling routine to
      if L /= null then                    --* traverse through Agenda in Reverse
         L := L.PREVIOUS;                  --* order.
      end if;
   end PREVIOUS;

   function LOOK4(X : in ITEM; L : in LIST) return LIST is
      L1 : LIST := L;
   begin
      while NON_EMPTY(L1) loop
         if L1.ELEMENT = X then
            return L1;
         end if;
         NEXT(L1);
      end loop;
      return null;
   end LOOK4;


   procedure ADD(X : in ITEM; L : in out LIST) is
   -- ITEM IS ADDED TO THE HEAD OF THE LIST
      T : LIST := new NODE;
   begin
      T.ELEMENT := X;
      T.PREVIOUS := null; --* (Apr 91)
      if L = null then
         T.NEXT := null;
      else
         T.NEXT := L;
       L.PREVIOUS := T; --* (Apr 91)
   end if;
   L := T;
end ADD;

function SUBSEQUENCE(L1 : in LIST; L2 : in LJST) return BOOLEAN is
   L : LIST := L1;
begin
   while NON_EMPTY(L) loop
      if not MEMBER(VALUE(L), L2) then
         return FALSE;
      end if;
      NEXT(L);
    end loop;
   return TRUE;
end SUBSEQUENCE;

function EQUAL(L1 : in LIST; L2 : in LIST) return BOOLEAN is
begin
   return (SUBSEQUENCE(L1, L2) and SUBSEQUENCE(L2, L1));
end EQUAL;

procedure EMPTY(L : out LIST) is
begin
    L := null;
end EMPTY;

function MEMBER(X : in ITEM; L : in LIST) return BOOLEAN is
begin
   if LOOK4(X, L) /= null then
      return TRUE;
   else
      return FALSE;
   end if;
end MEMBER;

procedure REMOVE(X : in ITEM; L : in out LIST) is
   HEADER, --* ADDED ON 21 MAY 91 TO CORRECT ERROR
    CURR : LIST := L;
      PREV : LIST := null;
      TEMP : LIST := null;
begin
   while NON_EMPTY(CURR) loop
       if VALUE(CURR) = X then
         TEMP := CURR;
         NEXT(CURR);
         TEMP.PREVIOUS := null;
         TEMP.NEXT := null;
         FREE(TEMP);
         if PREV /= null then --* Operator we are removing is within list
            PREV.NEXT := CURR;
         else
            HEADER := CURR; --* ADDED 21 MAY 91 TO CORRECT ERROR
         end if;
         if CURR /= null then --* List contains other items so we must relink
            CURR.PREVIOUS := PREV;--* the list in reverse. --* (Apr 91)
         end if;
       else
         PREV := CURR;
          NEXT(CURR);
      end if;
   end loop;
```

```
     if NON_EMPTY(HEADER) then --* How do we handle removal of first item in list?
       L := HEADER; --* ADDED 21 MAY 91 TO CORRECT ERROR
     else --* diagnostics 2 June 91
       L := CURR; --* diagnostics 2 June 91
     end if; --* diagnostics 2 June 91
end REMOVE;

  procedure LIST_REVERSE(L1 : in LIST; L2 : in out LIST) is
     L : LIST := L1;
  begin
     EMPTY(L2);
     while NON_EMPTY(L) loop
       ADD(VALUE(L), L2);
       NEXT(L);
     end loop;
  end LIST_REVERSE;

  procedure FREE_LIST(L: in out LIST) is
     CURR: LIST := L;
     TEMP: LIST;
   begin
     while NON_EMPTY(L) loop
       NEXT(CURR);
        if NON_EMPTY(CURR) then
          CURR.PREVIOUS := null;
       end if;
       L.NEXT := null;
       FREE(L);
       L:= CURR;
     end loop;
  end FREE_LIST;

  procedure DUPLICATE(L1 : in LIST; L2 : in out LIST) is
  TEMP : LIST := null;
  L : LIST := L1;
  begin
     FREE_LIST(L2);
     while NON_EMPTY(L) loop
       ADD(VALUE(L), TEMP);
       NEXT(L);
     end loop;
     LIST_REVERSE(TEMP, L2);
  end DUPLICATE;

   function VALUE(L : in LIST) return ITEM is
  begin
     if NON_EMPTY(L) then
       return L.ELEMENT;
     else
       raise BAD_VALUE;
     end if;
  end VALUE;

  procedure INSERT_NEXT(X : in ITEM; L : in out LIST) is
     T    : LIST := new NODE;
  begin
     T.ELEMENT := X;
     if NON_EMPTY(L) then
       if L.NEXT /= null then
          L.NEXT.PREVIOUS := T;
       end if;
```

```
         T.PREVIOUS := L;
         T.NEXT := L.NEXT;
         L.NEXT := T;
      end if;
      L := T;
   end INSERT_NEXT;

    procedure REPLACE_ITEM(X : in ITEM; L : in out LIST) is
   begin
      L.ELEMENT := X;
    end REPLACE_ITEM;

   procedure COPY_LIST(L1 : in LIST; L2 : in out LIST) is

      CURR: LIST := L1;
      HEAD: LIST := L2;
      TEMP: LIST;
      PREV: LIST;

    begin
      while NON_EMPTY(CURR) and NON_EMPTY(L2) loop
          L2.ELEMENT := VALUE(CURR);
          NEXT(CURR);
          PREV := L2;
          NEXT(L2);
       end loop;
 --* HANDLE CASE WHEN L2 IS LONGER THAN L1;
      if not NON_EMPTY(CURR) and NON_EMPTY(L2) then
         PREV.NEXT := null; --* DISCONNECT EXCESS FROM L2
         while NON_EMPTY(L2) loop
            TEMP := L2;
            TEMP.PREVIOUS := null;
            NEXT(L2);
            TEMP.NEXT := null;
            FREE(TEMP);
         end loop;
 --* HANDLE CASE WHEN L1 IS LONGER THAN L2;
      elsif NON_EMPTY(CURR) and not NON_EMPTY(L2) then
          while NON_EMPTY(CURR) loop
            TEMP := new NODE;
             PREV.NEXT := TEMP;
            TEMP.ELEMENT := VALUE(CURR);
            TEMP.PREVIOUS := PREV;
            PREV := TEMP;
            NEXT(CURR);
         end loop;
      end if;
      L2 := HEAD;
   end COPY_LIST;

end SEQUENCES;
```

```
with TEXT_IO;
with DATA;
with SEQUENCES;
with FRONT_END; use FRONT_END;

package TOP_SORT is

   procedure T_SORT(PRECEDENCE_LIST: in out DATA.NODE_LIST.LIST;
                    COUNT: in INTEGER);
    --* This procedure produces a topological sort of the operators that are in
    --* the NEW_GRAPH structure.

   end TOP_SORT;

   package body TOP_SORT is

    procedure T_SORT(PRECEDENCE_LIST     : in out DATA.NODE_LIST.LIST;
                    COUNT                : in INTEGER) is

    package int_io is new TEXT_IO.integer_io(integer);
    use int_io;

    type DEGREES is array (0..COUNT) of INTEGER;

    IN_DEGREE: DEGREES; --* Indegree Array used in sorting

    package QUEUES is new SEQUENCES(INTEGER);


    PARENT_LIST                  : DATA.NODE_LIST.LIST;
    CHILD_LIST                   : DATA.NODE_LIST.LIST;
    PARENT_COUNT                 : INTEGER;
    CHILD_COUNT                  : INTEGER;

    QUEUE                        : QUEUES.LIST;
    HEAD                         : QUEUES.LIST;

    REVERSED_PREC_LIST           : DATA.NODE_LIST.LIST;

 begin
   for OP in 1..COUNT loop
     FRONT_END.NEW_GRAPH.RETURN_PARENT_LIST(PARENT_LIST, OP, PARENT_COUNT);
     IN_DEGREE(OP) := PARENT_COUNT;
    end loop;
    QUEUES.ADD(0,QUEUE); --* BECAUSE OF THE USE OF A DUMMY START NODE THIS NODE
--* WILL ALWAYS BE THE FIRST ELEMENT IN THE QUEUE WITH
--* AN IN_DEGREE OF ZERO.
    HEAD := QUEUE;
    while QUEUES.NON_EMPTY(QUEUE) loop
      FRONT_END.NEW_GRAPH.RETURN_CHILD_LIST(CHILD_LIST, QUEUES.VALUE(HEAD),
              CHILD_COUNT);
         while DATA.NODE_LIST.NON_EMPTY(CHILD_LIST) loop
           IN_DEGREE(DATA.NODE_LIST.VALUE(CHILD_LIST)) :=
              IN_DEGREE(DATA.NODE_LIST.VALUE(CHILD_LIST)) - 1;
            if IN_DEGREE(DATA.NODE_LIST.VALUE(CHILD_LIST))= 0 then
               QUEUES.ADD(DATA.NODE_LIST.VALUE(CHILD_LIST), QUEUE);
            end if;
```

```
        DATA.NODE_LIST.NEXT(CHILD_LIST);
      end loop;
      DATA.NODE_LIST.ADD(QUEUES.VALUE(HEAD), REVERSED_PREC_LIST);
      QUEUES.REMOVE(QUEUES.VALUE(HEAD), QUEUE);
      HEAD := QUEUE;
    end loop;
    DATA.NODE_LIST.LIST_REVERSE(REVERSED_PREC_LIST, PRECEDENCE_LIST);
  end T_SORT;

end TOP_SORT;
```

```
with DATA; use DATA;

package PROCESSOR is

   procedure FIND_PERIODS(OP_LIST : in out V_LISTS.LIST);

   procedure VALIDATE_DATA (OP_LIST : in out V_LISTS.LIST);


   NOT_FEASIBLE                                   : exception;
   CRIT_OP_LACKS_MET                              : exception;
   MET_NOT_LESS_THAN_PERIOD                       : exception;
   MET_NOT_LESS_THAN_MRT                          : exception;
   MCP_NOT_LESS_THAN_MRT                          : exception;
   MCP_LESS_THAN_MET                              : exception;
   MET_IS_GREATER_THAN_FINISH_WITHIN              : exception;
   SPORADIC_OP_LACKS_MCP                          : exception;
   SPORADIC_OP_LACKS_MRT                          : exception;
   PERIOD_LESS_THAN_FINISH_WITHIN                 : exception;

end PROCESSOR;
```

```
with TEXT_IO;
with DATA; use DATA;

package body PROCESSOR is

   procedure FIND_PERIODS(OP_LIST : in out V_LISTS.LIST) is

   TARGET                  : V_LISTS.LIST;
   N                       : NATURAL := 0;
   L                       : FLOAT := 0.0;
   NEW_PERIOD              : NATURAL := 0;
   OP                      : OPERATOR;
   C                       : FLOAT;
   FIRST                   : BOOLEAN := true;
   FOUND                   : BOOLEAN := false;
   FRACTION                : NATURAL;
   FR_GCD                  : NATURAL;
   LCM                     : NATURAL;
   UNIT                    : NATURAL;
   ALPHA                   : FLOAT;
   GCD                     : NATURAL;

   package I_IO is new TEXT_IO.INTEGER_IO(NATURAL);

   procedure CALCULATE_NEW_PERIOD (O                  : in OPERATOR;
                                   NEW_PERIOD : in out NATURAL ) is
      DIFFERENCE : NATURAL;
      package VALUE_IO is new TEXT_IO.INTEGER_IO(NATURAL);
      begin
         DIFFERENCE := O.THE_MRT - O.THE_MET;
         if DIFFERENCE < O.THE_MCP then
            NEW_PERIOD := DIFFERENCE;
         else
            NEW_PERIOD := O.THE_MCP;
         end if;

   end CALCULATE_NEW_PERIOD;


   function FIND_GCD (SMALL : in VALUE; LARGE : in VALUE) return VALUE is
      REMAINDER : VALUE := SMALL;
   begin
       if LARGE mod SMALL = 0 then
         return REMAINDER;
      else
         REMAINDER := FIND_GCD(LARGE mod SMALL, SMALL);
         return REMAINDER;
      end if;
   end FIND_GCD;

   function FIND_LCM (NUMBER1, NUMBER2 : VALUE) return VALUE is
   begin
      return(NUMBER1 * NUMBER2) / GCD;
   end FIND_LCM;

   function REDUCE_TO_EVEN_FRACTION( GCD, PERIOD : NATURAL) return NATURAL is
      N : NATURAL := GCD / PERIOD;
```

```
begin
    if N * PERIOD = GCD then
        return N;
    else
        return N + 1;
    end if;
end REDUCE_TO_EVEN_FRACTION;

begin
-- FIRST PASS
-- Calculates the load factor for all periodic operators, and greatest common
-- divisor of the periods of the periodic operators
    TARGET := OP_LIST;
        while V_LISTS.NON_EMPTY(TARGET) loop
          OP := V_LISTS.VALUE(TARGET);
          if OP.THE_MET = 0 then
            Exception_Operator := OP.THE_OPERATOR_ID;
            raise CRIT_OP_LACKS_MET;
          elsif OP.THE_PERIOD /= 0 then -- a periodic operator
            L := L + FLOAT(OP.THE_MET)/FLOAT(OP.THE_PERIOD);
            if FIRST then
              GCD := OP.THE_PERIOD;
              FIRST := false;
            else
              if GCD > OP.THE_PERIOD then
                GCD := FIND_GCD(OP.THE_PERIOD,GCD);
              else
                GCD := FIND_GCD(GCD,OP.THE_PERIOD);
              end if;
            end if;
          end if;
      V_LISTS.NEXT(TARGET);
    end loop;

-- SECOND PASS
-- Finds the total number of sporadic operators (N)
-- For the sporadic opearators with user defined MRT or MCP values, calculates
-- the undefined value of MCP or MRT with given MRT or MCP
-- And finds the unit factor(UNIT) for the sporadic operators with user defined
-- MCP or MRT with calculated periods less than GCD found above

  TARGET := OP_LIST;
  FIRST := true;
  while V_LISTS.NON_EMPTY(TARGET) loop
   OP := V_LISTS.VALUE(TARGET);
   if OP.THE_PERIOD = 0 then -- a sporadic operator
      if OP.THE_MCP /= 0 and OP.THE_MRT = 0 then
        OP.THE_MRT := OP.THE_MET + OP.THE_MCP;
        TARGET.ELEMENT.THE_MRT := OP.THE_MRT;
     elsif OP.THE_MCP = 0 and OP.THE_MRT /= 0 then
        OP.THE_MCP := OP.THE_MRT - OP.THE_MET;
        TARGET.ELEMENT.THE_MCP := OP.THE_MCP;
     end if;
     if OP.THE_MCP /= 0 and OP.THE_MRT /= 0 then
        CALCULATE_NEW_PERIOD(OP,NEW_PERIOD);
        if NEW_PERIOD < GCD then
          FOUND := true;
          FRACTION := GCD/REDUCE_TO_EVEN_FRACTION(GCD,NEW_PERIOD);
            if FIRST then
              FR_GCD := FRACTION;
              LCM := FRACTION;
            FIRST := false;
          else
            if FRACTION > FR_GCD then
              FR_GCD := FIND_GCD(FR_GCD, FRACTION);
            else
              FR_GCD := FIND_GCD(FRACTION,FR_GCD);
            end if;
            LCM := FIND_LCM(LCM,FRACTION);
          end if;
      end if;
    else
      N := N + 1;
    end if;
  end if;
  V_LISTS.NEXT(TARGET);
end loop;
if FOUND then
  UNIT := LCM;
else
      UNIT := GCD;
end if;

-- THIRD PASS
-- Calculates and writes the periods for the sporadic operators with user defined
-- MCP or MRT by using the UNIT factor calculated above. Modifies the load factor
-- L calculated in the first pass. Finds coefficient ALPHA.


TARGET := OP_LIST;
    while V_LISTS.NON_EMPTY(TARGET) loop
      OP := V_LISTS.VALUE(TARGET);
      if OP.THE_PERIOD = 0 then -- a sporadic operator
        if OP.THE_MRT /= 0 and OP.THE_MCP /= 0 then
          CALCULATE_NEW_PERIOD(OP,NEW_PERIOD);
          NEW_PERIOD := NEW_PERIOD - NEW_PERIOD mod UNIT;
          OP.THE_PERIOD := NEW_PERIOD;
          TEXT_IO.PUT(“New PERIOD for operator “);
          VARSTRING.PUT(OP.THE_OPERATOR_ID);
          TEXT_IO.PUT(“ is “);
          I_IO.PUT(NEW_PERIOD,1);
          TEXT_IO.NEW_LINE;
          TARGET.ELEMENT.THE_PERIOD := OP.THE_PERIOD;
          L := L + FLOAT(OP.THE_MET)/FLOAT(NEW_PERIOD);
        end if;
      end if;
      V_LISTS.NEXT(TARGET);
    end loop;

    if L < 0.5 then
      C := 0.5;
    elsif L >= 0.5 and L < 1.0 then
      C := (1.0 + L) / 2.0;
    else
      raise NOT_FEASIBLE;
    end if;
    ALPHA := FLOAT(N)/(C - L) + 1.0;
    if ALPHA < 2.0 then
      ALPHA := 2.0;
    end if;

-- FOURTH PASS
```

```
  -- Calculates and writes the PERIOD, MRT, MCP values for the sporadic operators
  -- without user defined MCP or MRT values


  TARGET := OP_LIST;
  while V_LISTS.NON_EMPTY(TARGET) loop
    OP := V_LISTS.VALUE(TARGET);
      if OP.THE_PERIOD = 0 then -- a sporadic operator
        if OP.THE_MRT = 0 and OP.THE_MCP = 0 then
          OP.THE_MRT := NATURAL(ALPHA) * OP.THE_MET;
          OP.THE_MCP := OP.THE_MRT - OP.THE_MET;
        if (OP.THE_MCP / UNIT) * UNIT /= OP.THE_MCP then
          OP.THE_PERIOD := OP.THE_MCP + UNIT - (OP.THE_MCP mod UNIT);
        else
          OP.THE_PERIOD := OP.THE_MCP;
        end if;
        TEXT_IO.PUT(“New PERIOD for operator “);
        VARSTRING.PUT(OP.THE_OPERATOR_ID);
        TEXT_IO.PUT(“ is “);
        I_IO.PUT(OP.THE_PERIOD,1);
        TEXT_IO.NEW_LINE;
      end if;
    end if;
    TARGET.ELEMENT.THE_PERIOD := OP.THE_PERIOD;
    TARGET.ELEMENT.THE_MRT := OP.THE_MRT;
    TARGET.ELEMENT.THE_MCP := OP.THE_MCP;
    V_LISTS.NEXT(TARGET);
  end loop;
end FIND_PERIODS;


procedure VALIDATE_DATA (OP_LIST : in out V_LISTS.LIST) is

    TARGET : V_LISTS.LIST;
    package VAL_IO is new TEXT_IO.INTEGER_IO(VALUE);
begin
TARGET := OP_LIST;
    while V_LISTS.NON_EMPTY(TARGET) loop

-- ensure that there is no operator without an MET.
      if V_LISTS.VALUE(TARGET).THE_MET = 0 then
        Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
        raise CRIT_OP_LACKS_MET;
      end if;
      if V_LISTS.VALUE(TARGET).THE_PERIOD = 0 then
-- Check to ensure that MCP has a value for sporadic operators
        if V_LISTS.VALUE(TARGET).THE_MCP = 0 then
          Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
          raise SPORADIC_OP_LACKS_MCP;
        elsif V_LISTS.VALUE(TARGET).THE_MET >
          V_LISTS.VALUE(TARGET).THE_MCP then
          Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
          raise MCP_LESS_THAN_MET;
        end if;
-- Check to ensure that MRT has a value for sporadic operators
        if V_LISTS.VALUE(TARGET).THE_MRT = 0 then
          Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
          raise SPORADIC_OP_LACKS_MRT;
        end if;

-- Check to ensure that the MRT is greater than the MET.
```

```
               if V_LISTS.VALUE(TARGET).THE_MET > V_LISTS.VALUE(TARGET).THE_MRT then
                  Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
                  raise MET_NOT_LESS_THAN_MRT;
               end if;

     -- Guarantees that an operator can fire at least once
     -- before a response expected.
            if V_LISTS.VALUE(TARGET).THE_MCP > V_LISTS.VALUE(TARGET).THE_MRT then
               Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
               raise MCP_NOT_LESS_THAN_MRT;
            end if;

     else
     -- Check to ensure that the PERIOD is greater than the MET.
     if V_LISTS.VALUE(TARGET).THE_MET > V_LISTS.VALUE(TARGET).THE_PERIOD then
            Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
            raise MET_NOT_LESS_THAN_PERIOD;
     end if;

     -- Check to ensure that the FINISH_WITHIN is grater than the MET.
     if V_LISTS.VALUE(TARGET).THE_WITHIN /= 0 then
           if V_LISTS.VALUE(TARGET).THE_MET > V_LISTS.VALUE(TARGET).THE_WITHIN then
               Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
               raise MET_IS_GREATER_THAN_FINISH_WITHIN;
           elsif V_LISTS.VALUE(TARGET).THE_PERIOD <
                  V_LISTS.VALUE(TARGET).THE_WITHIN then
              Exception_Operator := V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
              raise PERIOD_LESS_THAN_FINISH_WITHIN;
           end if;
     end if;

    end if;

    V_LISTS.NEXT(TARGET);
   end loop;
  end VALIDATE_DATA;


end PROCESSOR;
```

APPENDIX D. NEW PACKAGES

```
with TEXT_IO;
with DATA; use DATA;


--* This package contains the specifications for a graph data structure that can
--* represent an acyclic graph. Functions and procedures exist to access the
--* information that is stored in the graph as well as to find out the relationships
--* between vertices in the graph.

generic

package NEW_DATA_STRUCTURES is

  type GRAPH (SIZE : INTEGER) is limited private;

  type GRAPH_LINK is access GRAPH;

  THE_GRAPH                                  : GRAPH_LINK;


 procedure PRODUCE_OP_ARRAY (INFO_LIST : in out V_LISTS.LIST;
                             COUNT       : in INTEGER);
 --* Transfer operator info from linked list to array

 function OP_POSITION (OP_NAME          : in VARSTRING.VSTRING;
                       COUNT            : in INTEGER) return INTEGER;
 --* Given an operator name return the operator's position in the array

 procedure PRODUCE_OP_MATRIX (COUNT: in INTEGER);
 --* Create a Matrix to represent the acyclic graph of operator relationship

 function OP_RETURN (OP_POSITION: in INTEGER) return OPERATOR;
 --* Given an operator's position in the array, return the operator

 function IS_PARENT (OP_1                   : in INTEGER;
                     OP_2                   : in INTEGER) return BOOLEAN;
 --* Return true if OP_1 is a parent of OP_2 or if OP_1 is OP_2

 function IS_CHILD (OP_1                    : in INTEGER;
                    OP_2                    : in INTEGER) return BOOLEAN;
 --* Return true if OP_1 is a child of OP_2 or if OP_1 is OP_2

 procedure RETURN_PARENT_LIST (PARENT_LIST     : in out NODE_LIST.LIST;
                               OP              : in INTEGER;
                               COUNT           : in out INTEGER);
 --* Return a list of all the parents of an operator

 procedure RETURN_CHILD_LIST   (CHILD_LIST     : in out NODE_LIST.LIST;
                                OP             : in INTEGER;
                                COUNT          : in out INTEGER);
 --* Return a list of all the children of an operator

 procedure FREE_GRAPH (A_GRAPH: in out GRAPH_LINK);
--* Free the memory space used by the graph
 private

 type INFO_ARRAY is array (INTEGER range <>) of OPERATOR;
type MATRIX_OP_INFO is
  record
   PARENT    : INTEGER := -1;
   CHILD     : INTEGER := -1;
end record;

type MATRIX is array (INTEGER range <>,INTEGER range <>) of MATRIX_OP_INFO;

type GRAPH (SIZE : INTEGER) is
   record
      OP_ARRAY           : INFO_ARRAY(0..SIZE);
     OP_MATRIX           : MATRIX(0..SIZE, 0..SIZE);
end record;


end NEW_DATA_STRUCTURES;
```

```
with UNCHECKED_DEALLOCATION;

package body NEW_DATA_STRUCTURES is

   pragma LINK_WITH ("heaplib.sparc.ar");

   procedure FREE is new UNCHECKED_DEALLOCATION(GRAPH, GRAPH_LINK);

   package int_io is new TEXT_IO.integer_io(integer);--put in for debugging
   use int_io;

   procedure PRODUCE_OP_ARRAY (INFO_LIST      : in out V_LISTS.LIST;
                               COUNT          : in INTEGER) is

     HEAD        : V_LISTS.LIST := INFO_LIST;

   function MAKE_START_NODE return OPERATOR is

     START_OP       : OPERATOR;

   begin
     START_OP.THE_OPERATOR_ID := VARSTRING.VSTR("DUMMY START NODE");
     START_OP.THE_MET := 0;
     START_OP.THE_MRT := 0;
     START_OP.THE_MCP := 0;
     START_OP.THE_WITHIN := 0;
     return START_OP;
   end MAKE_START_NODE;


   begin
     for INDEX in reverse 1..COUNT loop
       THE_GRAPH.OP_ARRAY(INDEX) := V_LISTS.VALUE(INFO_LIST);
       V_LISTS.NEXT(INFO_LIST);
     end loop;
     THE_GRAPH.OP_ARRAY(0) := MAKE_START_NODE;
     V_LISTS.FREE_LIST(HEAD); --* THIS LIST IS NO LONGER NEEDED.
   end PRODUCE_OP_ARRAY;

   function OP_POSITION   (OP_NAME    : in VARSTRING.VSTRING;
                           COUNT      : in INTEGER) return INTEGER is

   --* This function is implemented now as a linear scan. Its performance
   --* can be improved by using a hashing function. If a hashing function
   --* is to be used, then the procedure PRODUCE_OP_ARRAY will also have
   --* to be modified if hashing is to be used. 17 July 91

   begin
     for INDEX in 1..COUNT loop
       if VARSTRING.EQUAL (OP_NAME,
         THE_GRAPH.OP_ARRAY(INDEX).THE_OPERATOR_ID) then
         return INDEX;
       end if;
     end loop;
     return -1; --* Operator is external since it is not in the array.
   end OP_POSITION;
```

```
procedure PRODUCE_OP_MATRIX (COUNT          : in INTEGER) is

  COLUMN,
  ROW,
  PARENT_OP,
  CHILD_OP                  : INTEGER;

  LINK                      : constant VARSTRING.VSTRING := VARSTRING.VSTR("LINK");

    procedure INITIALIZE (COUNT             : in INTEGER;
                          OP_MATRIX         : in out MATRIX) is

    begin
      for ROW in 0..COUNT loop
        for COLUMN in 0..COUNT loop
          if ROW = COLUMN then
            THE_GRAPH.OP_MATRIX(ROW,COLUMN).PARENT := ROW;
            THE_GRAPH.OP_MATRIX(ROW,COLUMN).CHILD := ROW;
          end if;
        end loop;
      end loop;
    end INITIALIZE;

procedure INITIALIZE_START_NODE (COUNT                : in INTEGER;
                                 OP_MATRIX            : in out MATRIX) is

begin
  for INDEX in 0..COUNT loop
    if THE_GRAPH.OP_MATRIX(INDEX, INDEX).PARENT = INDEX then
      THE_GRAPH.OP_MATRIX(INDEX,INDEX).PARENT := 0;
      THE_GRAPH.OP_MATRIX(0,INDEX).CHILD := THE_GRAPH.OP_MATRIX(0,0).CHILD;
      THE_GRAPH.OP_MATRIX(0,0).CHILD := INDEX;
      THE_GRAPH.OP_MATRIX(0,INDEX).PARENT := INDEX;
    end if;
  end loop;
end INITIALIZE_START_NODE;



begin
  TEXT_IO.OPEN (AG_OUTFILE,INPUT,"atomic.info");
  INITIALIZE(COUNT, THE_GRAPH.OP_MATRIX);
  VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
  while not TEXT_IO.END_OF_FILE(AG_OUTFILE) loop
    if VARSTRING.EQUAL (New_Word,LINK) then -- keyword "LINK"
      TEXT_IO.SKIP_LINE(AG_OUTFILE); -- skip LINK name
      VARSTRING.GET_LINE(AG_OUTFILE, New_Word);
      PARENT_OP := OP_POSITION(New_Word, DATA.OP_COUNT);
      TEXT_IO.SKIP_LINE(AG_OUTFILE);
      VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
      CHILD_OP := OP_POSITION(New_Word, DATA.OP_COUNT);


-- when either starting node or ending node of a link is EXTERNAL,
-- the link information will not be added to the graph. Assuming
-- that all external data coming in is ready at start time.

      if PARENT_OP /= -1 and CHILD_OP /= -1 then
        THE_GRAPH.OP_MATRIX(PARENT_OP,CHILD_OP).CHILD :=
           THE_GRAPH.OP_MATRIX(PARENT_OP,PARENT_OP).CHILD;
        THE_GRAPH.OP_MATRIX(PARENT_OP,PARENT_OP).CHILD := CHILD_OP;
        THE_GRAPH.OP_MATRIX(PARENT_OP,CHILD_OP).PARENT :=
          THE_GRAPH.OP_MATRIX(CHILD_OP,CHILD_OP).PARENT;
        THE_GRAPH.OP_MATRIX(CHILD_OP,CHILD_OP).PARENT := PARENT_OP;
      end if;
      VARSTRING.GET_LINE ( AG_OUTFILE, New_Word);
    else
      VARSTRING.GET_LINE ( AG_OUTFILE, New_Word); -- skip all other info
    end if;
  end loop;
  TEXT_IO.CLOSE (AG_OUTFILE);
  INITIALIZE_START_NODE(COUNT, THE_GRAPH.OP_MATRIX);
end PRODUCE_OP_MATRIX;

function OP_RETURN (OP_POSITION: in INTEGER) return OPERATOR is

  OP    : OPERATOR;

begin
  OP := THE_GRAPH.OP_ARRAY(OP_POSITION);
  return OP;
end OP_RETURN;

function IS_PARENT (OP_1            : in INTEGER;
                    OP_2            : in INTEGER) return BOOLEAN is

--* Return true if OP_1 is a parent of OP_2 or if OP_1 is OP_2

  PARENT                : BOOLEAN := false;

begin
  if OP_1 = OP_2 then
    PARENT := true;
  elsif THE_GRAPH.OP_MATRIX(OP_1, OP_2).PARENT /= -1 then
     PARENT := true;
  end if;
  return PARENT;
end IS_PARENT;

function IS_CHILD (OP_1             : in INTEGER;
                   OP_2             : in INTEGER) return BOOLEAN is

--* Return true if OP_1 is a child of OP_2 or if OP_1 is OP_2

CHILD    : BOOLEAN := false;

begin
  if OP_1 = OP_2 then
     CHILD := true;
  elsif THE_GRAPH.OP_MATRIX(OP_2, OP_1).CHILD /= -1 then
    CHILD := true;
  end if;
  return CHILD;
end IS_CHILD;

procedure RETURN_PARENT_LIST (PARENT_LIST           : in out NODE_LIST.LIST;
                              OP                    : in INTEGER;
  COUNT          : in out INTEGER) is

  ROW            : INTEGER := OP;
```

```
begin
   COUNT := 0;
   while THE_GRAPH.OP_MATRIX(ROW, OP).PARENT /= OP loop
     NODE_LIST.ADD(THE_GRAPH.OP_MATRIX(ROW, OP).PARENT, PARENT_LIST);
     COUNT := COUNT + 1;
     ROW := THE_GRAPH.OP_MATRIX(ROW, OP).PARENT;
   end loop;
end RETURN_PARENT_LIST;

procedure RETURN_CHILD_LIST (CHILD_LIST      : in out NODE_LIST.LIST;
                             OP              : in INTEGER;
                             COUNT           : in out INTEGER) is
COLUMN          : INTEGER := OP;

begin
  COUNT := 0;
   while THE_GRAPH.OP_MATRIX(OP, COLUMN).CHILD /= OP loop
     NODE_LIST.ADD(THE_GRAPH.OP_MATRIX(OP, COLUMN).CHILD, CHILD_LIST);
     COUNT := COUNT + 1;
     COLUMN := THE_GRAPH.OP_MATRIX(OP, COLUMN).CHILD;
  end loop;
end RETURN_CHILD_LIST;

procedure FREE_GRAPH (A_GRAPH: in out GRAPH_LINK) is

begin
  FREE(A_GRAPH);
end FREE_GRAPH;

end NEW_DATA_STRUCTURES;
```

```
with TEXT_IO;
with DATA; use DATA;
with NEW_DATA_STRUCTURES;

package FRONT_END is

   procedure PRODUCE_OP_LIST(INFO_LIST          : in out V_LISTS.LIST;
                                       COUNT               : in out INTEGER);
   --* Extract the operator information from the ATOMIC.INFO file
   --* and place it in a linked list.

   procedure TEST_DATA (INPUT_LIST                              : in V_LISTS.LIST;
                                HARMONIC_BLOCK_LENGTH        : in INTEGER);
   --* Determine if the operators can be feasabily scheduled on a single
   --* processor system.

   package NEW_GRAPH is new NEW_DATA_STRUCTURES;
   --* Instantiate the graph data structure so that it can be accessed by
   --* the rest of the Static Scheduler.

   NUMBER_OF_OPERATORS : INTEGER;

end FRONT_END;

  package body FRONT_END is

   procedure PRODUCE_OP_LIST (INFO_LIST         : in out V_LISTS.LIST;
                                         COUNT             : in out INTEGER) is

   -- This procedure reads the output file which has the link information with
   -- the Atomic operators and depending upon the keywords that are declared
   -- as constants separates the information in the file and stores it the
   -- operator array and the link matrix.

   package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);

   MET          : constant VARSTRING.VSTRING := VARSTRING.VSTR("MET");
   MRT          : constant VARSTRING.VSTRING := VARSTRING.VSTR("MRT");
   MCP          : constant VARSTRING.VSTRING := VARSTRING.VSTR("MCP");
   PERIOD     : constant VARSTRING.VSTRING := VARSTRING.VSTR("PERIOD");
   WITHIN     : constant VARSTRING.VSTRING := VARSTRING.VSTR("WITHIN");
   LINK         : constant VARSTRING.VSTRING := VARSTRING.VSTR("LINK");
   ATOMIC    : constant VARSTRING.VSTRING := VARSTRING.VSTR("ATOMIC");
   EMPTY      : constant VARSTRING.VSTRING := VARSTRING.VSTR("EMPTY");



   procedure INITIALIZE_OPERATOR (OP : in out OPERATOR) is
   begin
      OP.THE_MET := 0;
      OP.THE_MRT := 0;
      OP.THE_MCP := 0;
      OP.THE_PERIOD := 0;
      OP.THE_WITHIN := 0;
   end INITIALIZE_OPERATOR;
   begin
     TEXT_IO.OPEN (AG_OUTFILE,INPUT,”atomic.info”);
     TEXT_IO.CREATE(NON_CRITS,OUTPUT,”non_crits”);
     COUNT := 0;
     VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
      while not TEXT_IO.END_OF_FILE(AG_OUTFILE) loop
        if VARSTRING.EQUAL (New_Word,LINK) then -- keyword “LINK”
           TEXT_IO.SKIP_LINE(AG_OUTFILE); -- Skip over LINK
           TEXT_IO.SKIP_LINE(AG_OUTFILE); -- info for now.
           TEXT_IO.SKIP_LINE(AG_OUTFILE);
           TEXT_IO.SKIP_LINE(AG_OUTFILE);
           VARSTRING.GET_LINE ( AG_OUTFILE, New_Word);
        elsif VARSTRING.EQUAL (New_Word,ATOMIC) then -- keyword “ATOMIC”
           VARSTRING.GET_LINE ( AG_OUTFILE, New_Word);
           Cur_Opt.THE_OPERATOR_ID := New_Word;
           VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
              if (VARSTRING.EQUAL(New_Word, ATOMIC)) or
                 (VARSTRING.EQUAL(New_Word, LINK)) or
                     (TEXT_IO.END_OF_FILE(AG_OUTFILE)) then
                 VARSTRING.PUT_LINE(NON_CRITS, Cur_Opt.THE_OPERATOR_ID);
   --* Non-periodic Operator, No need to be statically scheduled.
        else
           while VARSTRING.NOTEQUAL (New_Word, ATOMIC) and -- Loop to get
              VARSTRING.NOTEQUAL (New_Word, LINK) and -- timing info
                 not TEXT_IO.END_OF_FILE(AG_OUTFILE) loop -- of operator

              if VARSTRING.EQUAL (New_Word,MET) then -- keyword “MET”
                 VALUE_IO.GET(AG_OUTFILE,Current_Value);
                 TEXT_IO.SKIP_LINE(AG_OUTFILE);
                 Cur_Opt.THE_MET := Current_Value;
              elsif VARSTRING.EQUAL (New_Word,MRT) then -- keyword “MRT”
                 VALUE_IO.GET(AG_OUTFILE,Current_Value);
                 TEXT_IO.SKIP_LINE(AG_OUTFILE);
                 Cur_Opt.THE_MRT:= Current_Value;
              elsif VARSTRING.EQUAL (New_Word,MCP) then -- keyword “MCP”
                 VALUE_IO.GET(AG_OUTFILE,Current_Value);
                 TEXT_IO.SKIP_LINE(AG_OUTFILE);
                  Cur_Opt.THE_MCP := Current_Value ;
              elsif VARSTRING.EQUAL (New_Word,PERIOD) then -- keyword “PERIOD”
                 VALUE_IO.GET(AG_OUTFILE,Current_Value);
                 TEXT_IO.SKIP_LINE(AG_OUTFILE);
                 Cur_Opt.THE_PERIOD := Current_Value;
               elsif VARSTRING.EQUAL (New_Word,WITHIN) then -- keyword “WITHIN”
                 VALUE_IO.GET(AG_OUTFILE,Current_Value);
                 TEXT_IO.SKIP_LINE(AG_OUTFILE);
                  Cur_Opt.THE_WITHIN := Current_Value;
               end if;
               VARSTRING.GET_LINE(AG_OUTFILE,New_Word);
           end loop;
           V_LISTS.ADD(Cur_Opt, INFO_LIST);
           COUNT := COUNT + 1;
          INITIALIZE_OPERATOR(Cur_Opt);
        end if;
      end if;
   end loop;
   TEXT_IO.CLOSE(AG_OUTFILE);
   NUMBER_OF_OPERATORS := COUNT;
end PRODUCE_OP_LIST;
```

```
procedure TEST_DATA (INPUT_LIST                               : in V_LISTS.LIST;
                     HARMONIC_BLOCK_LENGTH                    : in INTEGER) is

procedure CALC_TOTAL_TIME (INPUT_LIST                         : in V_LISTS.LIST;
                           HARMONIC_BLOCK_LENGTH : in INTEGER) is
  V                    : V_LISTS.LIST := INPUT_LIST;
  TIMES                : FLOAT := 0.0;
  OP_TIME              : FLOAT := 0.0;
  TOTAL_TIME           : FLOAT := 0.0;
  PER                  : OPERATOR;
  BAD_TOTAL_TIME       : exception;

  function CALC_NO_OF_PERIODS (HARMONIC_BLOCK_LENGTH : in INTEGER;
                               THE_PERIOD        : in INTEGER) return FLOAT is
  begin
    return FLOAT(HARMONIC_BLOCK_LENGTH) / FLOAT(THE_PERIOD);
  end CALC_NO_OF_PERIODS;

function MULTIPLY_BY_MET (TIMES : in FLOAT;
  THE_MET : in VALUE) return FLOAT is
begin
  return TIMES * FLOAT(THE_MET);
end MULTIPLY_BY_MET;

function ADD_TO_SUM (OP_TIME : in FLOAT) return FLOAT is
begin
  return TOTAL_TIME + OP_TIME;
end ADD_TO_SUM;

begin --main CALC_TOTAL_TIME
  while V_LISTS.NON_EMPTY(V) loop
    PER := V_LISTS.VALUE(V);
    TIMES:= CALC_NO_OF_PERIODS (HARMONIC_BLOCK_LENGTH , PER.THE_PERIOD);
    OP_TIME := MULTIPLY_BY_MET (TIMES, V_LISTS.VALUE(V).THE_MET);
    TOTAL_TIME := ADD_TO_SUM (OP_TIME);
    if TOTAL_TIME > FLOAT(HARMONIC_BLOCK_LENGTH) then
       raise BAD_TOTAL_TIME;
    else
      V_LISTS.NEXT(V);
    end if;
  end loop;

exception
  when BAD_TOTAL_TIME =>
    TEST_VERIFIED := FALSE;
    TEXT_IO.PUT("The total execution time of the operators exceeds ");
    TEXT_IO.PUT_LINE("the HARMONIC_BLOCK_LENGTH");
    TEXT_IO.NEW_LINE;
end CALC_TOTAL_TIME;

procedure CALC_HALF_PERIODS (INPUT_LIST : in V_LISTS.LIST) is

  V : V_LISTS.LIST := INPUT_LIST;
  HALF_PERIOD : FLOAT;
  FAIL_HALF_PERIOD : exception;

function DIVIDE_PERIOD_BY_2 (THE_PERIOD : in VALUE) return FLOAT is
begin
  return FLOAT(THE_PERIOD) / 2.0;
end DIVIDE_PERIOD_BY_2;
```

```
begin --main CALC_HALF_PERIODS;
  while V_LISTS.NON_EMPTY(V) loop
    HALF_PERIOD := DIVIDE_PERIOD_BY_2(V_LISTS.VALUE(V).THE_PERIOD);
    if FLOAT(V_LISTS.VALUE(V).THE_MET) > HALF_PERIOD then
      Exception_Operator := V_LISTS.VALUE(V).THE_OPERATOR_ID;
      raise FAIL_HALF_PERIOD;
    else
      V_LISTS.NEXT(V);
    end if;
  end loop;

exception
  when FAIL_HALF_PERIOD =>
    TEST_VERIFIED := FALSE;
    TEXT_IO.PUT ("The MET of Operator ");
    VARSTRING.PUT (Exception_Operator);
    TEXT_IO.PUT_LINE (" is greater than half of its period.");
end CALC_HALF_PERIODS;

procedure CALC_RATIO_SUM (INPUT_LIST : in V_LISTS.LIST) is
  V                       : V_LISTS.LIST := INPUT_LIST;
  RATIO                   : FLOAT;
  RATIO_SUM               : FLOAT := 0.0;
  THE_MET                 : VALUE;
  THE_PERIOD              : VALUE;
  RATIO_TOO_BIG           : exception;

function DIVIDE_MET_BY_PERIOD (THE_MET : in VALUE;
  THE_PERIOD : in VALUE) return FLOAT is
begin
  return FLOAT(THE_MET) / FLOAT(THE_PERIOD);
end DIVIDE_MET_BY_PERIOD;

function ADD_TO_TIME (RATIO : in FLOAT) return FLOAT is
begin
  return RATIO_SUM + RATIO;
end ADD_TO_TIME;

begin --main CALC_RATIO_SUM
  while V_LISTS.NON_EMPTY(V) loop
    THE_MET := V_LISTS.VALUE(V).THE_MET;
    THE_PERIOD := V_LISTS.VALUE(V).THE_PERIOD;
    RATIO := DIVIDE_MET_BY_PERIOD(THE_MET,THE_PERIOD);
    RATIO_SUM := ADD_TO_TIME(RATIO);
    V_LISTS.NEXT(V);
  end loop;
  if RATIO_SUM - 0.5 > 0.00000001 then
    raise RATIO_TOO_BIG;
  end if;

exception
  when RATIO_TOO_BIG =>
    TEST_VERIFIED := FALSE;
    TEXT_IO.PUT ("The total MET/PERIOD ratio sum of operators is ");
    TEXT_IO.PUT_LINE ("greater than 0.5");
  end CALC_RATIO_SUM;

begin --main TEST_DATA
  CALC_TOTAL_TIME(INPUT_LIST, HARMONIC_BLOCK_LENGTH);
   CALC_HALF_PERIODS(INPUT_LIST);
```

```
      CALC_RATIO_SUM(INPUT_LIST);
    end TEST_DATA;

end FRONT_END;
```

```
--* This package is a generic priority queue. It requires three parameters to be
--* instantiated: A type of element to be stored in the priority queue, a value
--* to order the queue by, and a function to order the queue with.

generic
 type ELEMENT_1 is private;
 type ELEMENT_2 is private;
 with function ORDER_QUEUE (VALUE_1 : in ELEMENT_2;
                            VALUE_2 : in ELEMENT_2) return BOOLEAN;

package PRIORITY_QUEUES is

   type NODE;
   type LINK is access NODE;
   type NODE is
   record
      CONTENT                  : ELEMENT_1;
      VALUE                    : ELEMENT_2;
      NEXT                     : LINK;
   end record;

   function INITIALIZE_PRIORITY_QUEUE return LINK;

   procedure INSERT_IN_PRIORITY_QUEUE (ITEM          : in ELEMENT_1;
                                       ORDER_VALUE   : in ELEMENT_2;
                                       QUEUE         : in out LINK);

   function READ_BEST_FROM_PRIORITY_QUEUE (L: in LINK) return ELEMENT_1;
   --* This fuction reads the head of the queue without removing the item

   procedure REMOVE_BEST_FROM_PRIORITY_QUEUE (L: in out LINK);

   function NON_EMPTY(L : in LINK) return BOOLEAN;

end PRIORITY_QUEUES;

with UNCHECKED_DEALLOCATION;

package body PRIORITY_QUEUES is

   procedure FREE is new UNCHECKED_DEALLOCATION(NODE, LINK);

   function INITIALIZE_PRIORITY_QUEUE return LINK is

   L                           : LINK := null;
   begin
     return L;
   end INITIALIZE_PRIORITY_QUEUE;

   procedure INSERT_IN_PRIORITY_QUEUE (ITEM: in ELEMENT_1;
      ORDER_VALUE              : in ELEMENT_2;
      QUEUE                    : in out LINK) is
      FRONT                    : LINK := QUEUE;
      PREVIOUS                 : LINK := null;
      T                        : LINK := new NODE;
      OP_INSERTED              : BOOLEAN := false;
      NEW_FRONT                : BOOLEAN := true;
   begin
     T.CONTENT := ITEM;
     T.VALUE := ORDER_VALUE;
     while QUEUE /= null loop
       if ORDER_QUEUE(ORDER_VALUE, QUEUE.VALUE) then
           if PREVIOUS /= null then
             PREVIOUS.NEXT := T;
          end if;
           T.NEXT := QUEUE;
           OP_INSERTED := true;
          exit;
       end if;
       PREVIOUS := QUEUE;
       NEW_FRONT := false;
       QUEUE := QUEUE.NEXT;
     end loop;
     if not OP_INSERTED and FRONT /= null then
       PREVIOUS.NEXT := T;
     end if;
     if NEW_FRONT then
       QUEUE := T;
     else
       QUEUE := FRONT;
     end if;
   end INSERT_IN_PRIORITY_QUEUE;

   function READ_BEST_FROM_PRIORITY_QUEUE (L : in LINK) return ELEMENT_1 is

     BEST           : ELEMENT_1;

   begin
     BEST := L.CONTENT;
     return BEST;
   end READ_BEST_FROM_PRIORITY_QUEUE;


   procedure REMOVE_BEST_FROM_PRIORITY_QUEUE (L: in out LINK) is

     TEMP: LINK := L;

   begin
     L := L.NEXT;
     FREE(TEMP);
   end REMOVE_BEST_FROM_PRIORITY_QUEUE;

   function NON_EMPTY(L : in LINK) return BOOLEAN is
   begin
     if L = null then
       return FALSE;
     else
       return TRUE;
     end if;
   end NON_EMPTY;

end PRIORITY_QUEUES;
```

```
with DATA; use DATA;

package SCHEDULER is

 procedure EARLIEST_START(TOP_SORT              : in NODE_LIST.LIST;
                          AGENDA                : in out SCHEDULE_INPUTS_LIST.LIST;
                          COUNT                 : in INTEGER;
                          H_B_LENGTH            : in INTEGER;
                          VALID_SCHEDULE        : in out BOOLEAN);

 procedure EARLIEST_DEADLINE(TOP_SORT              : in NODE_LIST.LIST;
                             AGENDA                : in out SCHEDULE_INPUTS_LIST.LIST;
                             COUNT                 : in INTEGER;
                             H_B_LENGTH            : in INTEGER;
                             VALID_SCHEDULE        : in out BOOLEAN);

 procedure EXHAUSTIVE_ENUMERATION ( TOP_SORT: in NODE_LIST.LIST;
                                    AGENDA : in out SCHEDULE_INPUTS_LIST.LIST;
                                    OP_COUNT            : in INTEGER;
                                    H_B_LENGTH          : in INTEGER;
                                    VALID_SCHEDULE      : in out BOOLEAN);

 procedure CREATE_STATIC_SCHEDULE (OPERATOR_LIST : in NODE_LIST.LIST;
                          THE_SCHEDULE_INPUTS : in SCHEDULE_INPUTS_LIST.LIST;
                          HARMONIC_BLOCK_LENGTH : in INTEGER);

end SCHEDULER;
```

```
with TEXT_IO;
with DATA; use DATA;
with SEQUENCES;
with FRONT_END; use FRONT_END;
with PRIORITY_QUEUES;
with DIAGNOSTICS;

package body SCHEDULER is

    procedure EARLIEST_START(TOP_SORT                  : in NODE_LIST.LIST;
                                              AGENDA                    : in out SCHEDULE_INPUTS_LIST.LIST;
                                              COUNT                     : in INTEGER;
                                              H_B_LENGTH                : in INTEGER;
                                              VALID_SCHEDULE            : in out BOOLEAN) is

    package int_io is new TEXT_IO.integer_io(integer);
    use int_io;

    package EST_PRIORITY_QUEUES is new PRIORITY_QUEUES(DATA.SCHEDULE_INPUTS,
                                                                                  DATA.LOWERS,
                                                                                  "<");

       PRIORITY_QUEUE             : EST_PRIORITY_QUEUES.LINK := null;
       REV_AGENDA                 : SCHEDULE_INPUTS_LIST.LIST;
       T_SORT                     : NODE_LIST.LIST := TOP_SORT;
       NEW_NODE                   : SCHEDULE_INPUTS;
       BEST_NODE                  : SCHEDULE_INPUTS;
       ADDL_NODE                  : SCHEDULE_INPUTS;
       STOP_TIME                  : INTEGER := 0;
       OP_NUM                     : INTEGER;
       EST                        : INTEGER;
       TEMP                       : OPERATOR;


    begin
      VALID_SCHEDULE := true;
      NEW_NODE.THE_OPERATOR := 0;
      NEW_NODE.THE_LOWER := H_B_LENGTH + 10;
       SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, REV_AGENDA);
       NEW_NODE.THE_LOWER := 0;
       NODE_LIST.NEXT(T_SORT);
       while NODE_LIST.NON_EMPTY(T_SORT) or
            EST_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) loop
         if NODE_LIST.NON_EMPTY(T_SORT) then
            OP_NUM := DATA.NODE_LIST.VALUE(T_SORT);
             TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
            NEW_NODE.THE_OPERATOR := OP_NUM;
          end if;
          if EST_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) then
            BEST_NODE := EST_PRIORITY_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
         end if;
      if BEST_NODE.THE_LOWER < STOP_TIME and EST_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) then
          NEW_NODE.THE_OPERATOR := BEST_NODE.THE_OPERATOR;
         NEW_NODE.THE_LOWER := BEST_NODE.THE_LOWER;
          NEW_NODE.THE_START := STOP_TIME;
          TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
          STOP_TIME := STOP_TIME + TEMP.THE_MET;
         NEW_NODE.THE_STOP := STOP_TIME;
       NEW_NODE .THE_INSTANCE := BEST_NODE.THE_INSTANCE + 1;
       EST := NEW_NODE.THE_LOWER + TEMP.THE_PERIOD;
         if EST + TEMP.THE_MET <= H_B_LENGTH then
         ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
         ADDL_NODE.THE_LOWER := EST;
         ADDL_NODE.THE_INSTANCE := NEW_NODE.THE_INSTANCE;
         EST_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_LOWER, PRIORITY_QUEUE);
         end if;
       EST_PRIORITY_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
    elsif not NODE_LIST.NON_EMPTY(T_SORT) then
       NEW_NODE.THE_OPERATOR := BEST_NODE.THE_OPERATOR;
       NEW_NODE.THE_LOWER := BEST_NODE.THE_LOWER;
         if NEW_NODE.THE_LOWER > STOP_TIME then
            NEW_NODE.THE_START := NEW_NODE.THE_LOWER;
       else
            NEW_NODE.THE_START := STOP_TIME;
         end if;
       TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
       STOP_TIME := NEW_NODE.THE_START + TEMP.THE_MET;
       NEW_NODE.THE_STOP := STOP_TIME;
       NEW_NODE .THE_INSTANCE := BEST_NODE.THE_INSTANCE + 1;
       EST := NEW_NODE.THE_LOWER + TEMP.THE_PERIOD;
       if EST + TEMP.THE_MET <= H_B_LENGTH then
           ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
           ADDL_NODE.THE_LOWER := EST;
            ADDL_NODE.THE_INSTANCE := NEW_NODE.THE_INSTANCE;
            EST_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_LOWER, PRIORITY_QUEUE);
         end if;
       EST_PRIORITY_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
    else --* Scheduling Initial Set of Operators
       NEW_NODE.THE_START := STOP_TIME;
         TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
         STOP_TIME := STOP_TIME + TEMP.THE_MET;
         NEW_NODE.THE_STOP := STOP_TIME;
       NEW_NODE.THE_INSTANCE := 1;
       EST := NEW_NODE.THE_START + TEMP.THE_PERIOD;
        if EST + TEMP.THE_MET <= H_B_LENGTH or else NEW_NODE.THE_START >= TEMP.THE_PERIOD then
           ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
           ADDL_NODE.THE_LOWER := EST;
           ADDL_NODE.THE_INSTANCE := 1;
           EST_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_LOWER, PRIORITY_QUEUE);
       end if;
       NODE_LIST.NEXT(T_SORT);
   end if;
   if NEW_NODE.THE_STOP > H_B_LENGTH then
       VALID_SCHEDULE := false;
   end if;
   SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, REV_AGENDA);
   NEW_NODE.THE_LOWER := 0;
end loop;
SCHEDULE_INPUTS_LIST.LIST_REVERSE(REV_AGENDA, AGENDA);
SCHEDULE_INPUTS_LIST.FREE_LIST(REV_AGENDA);
end EARLIEST_START;

procedure EARLIEST_DEADLINE(TOP_SORT        : in NODE_LIST.LIST;
                            AGENDA          : in out SCHEDULE_INPUTS_LIST.LIST;
                            COUNT           : in INTEGER;
                            H_B_LENGTH      : in INTEGER;
                            VALID_SCHEDULE  : in out BOOLEAN) is
```

```
package int_io is new TEXT_IO.integer_io(integer);
use int_io;

package EDL_PRIORITY_QUEUES is new PRIORITY_QUEUES(DATA.SCHEDULE_INPUTS,
                                                   DATA.UPPERS,
                                                   “<“);

  PRIORITY_QUEUE            : EDL_PRIORITY_QUEUES.LINK := null;
  REV_AGENDA                : SCHEDULE_INPUTS_LIST.LIST;
  T_SORT                    : NODE_LIST.LIST := TOP_SORT;
  NEW_NODE                  : SCHEDULE_INPUTS;
  BEST_NODE                 : SCHEDULE_INPUTS;
  ADDL_NODE                 : SCHEDULE_INPUTS;
  STOP_TIME                 : INTEGER := 0;
  OP_NUM                    : INTEGER;
  EST                       : INTEGER;
  TEMP                      : OPERATOR;


begin
  VALID_SCHEDULE := true;
  NEW_NODE.THE_OPERATOR := 0;
  NEW_NODE.THE_LOWER := H_B_LENGTH + 10;
  SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, REV_AGENDA);
  NEW_NODE.THE_LOWER := 0;
  NODE_LIST.NEXT(T_SORT);
  while NODE_LIST.NON_EMPTY(T_SORT) or EDL_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) loop
    if NODE_LIST.NON_EMPTY(T_SORT) then
      OP_NUM := DATA.NODE_LIST.VALUE(T_SORT);
       TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
       NEW_NODE.THE_OPERATOR := OP_NUM;
    end if;
    if EDL_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) then
      BEST_NODE := EDL_PRIORITY_QUEUES.READ_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
    end if;
    if BEST_NODE.THE_LOWER < STOP_TIME and EDL_PRIORITY_QUEUES.NON_EMPTY(PRIORITY_QUEUE) then
      NEW_NODE.THE_OPERATOR := BEST_NODE.THE_OPERATOR;
      NEW_NODE.THE_LOWER := BEST_NODE.THE_LOWER;
      NEW_NODE.THE_UPPER := BEST_NODE.THE_UPPER;
      NEW_NODE.THE_START := STOP_TIME;
      TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
      STOP_TIME := STOP_TIME + TEMP.THE_MET;
      NEW_NODE.THE_STOP := STOP_TIME;
      NEW_NODE .THE_INSTANCE := BEST_NODE.THE_INSTANCE + 1;
      EST := NEW_NODE.THE_LOWER + TEMP.THE_PERIOD;
      if EST + TEMP.THE_MET <= H_B_LENGTH then
        ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
        ADDL_NODE.THE_LOWER := EST;
        ADDL_NODE.THE_INSTANCE := NEW_NODE.THE_INSTANCE;
        if TEMP.THE_WITHIN /= 0 then
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_WITHIN - TEMP.THE_MET;
        else
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_PERIOD - TEMP.THE_MET;
        end if;
        EDL_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_UPPER, PRIORITY_QUEUE);
       end if;
      EDL_PRIORITY_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
     elsif not NODE_LIST.NON_EMPTY(T_SORT) then
      NEW_NODE.THE_OPERATOR := BEST_NODE.THE_OPERATOR;
      NEW_NODE.THE_LOWER := BEST_NODE.THE_LOWER;
      NEW_NODE.THE_UPPER := BEST_NODE.THE_UPPER;
      if NEW_NODE.THE_LOWER > STOP_TIME then
        NEW_NODE.THE_START := NEW_NODE.THE_LOWER;
      else
        NEW_NODE.THE_START := STOP_TIME;
      end if;
      TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
      STOP_TIME := NEW_NODE.THE_START + TEMP.THE_MET;
      NEW_NODE.THE_STOP := STOP_TIME;
      NEW_NODE .THE_INSTANCE := BEST_NODE.THE_INSTANCE + 1;
       EST := NEW_NODE.THE_LOWER + TEMP.THE_PERIOD;
       if EST + TEMP.THE_MET <= H_B_LENGTH then
        ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
        ADDL_NODE.THE_LOWER := EST;
        ADDL_NODE.THE_INSTANCE := NEW_NODE.THE_INSTANCE;
        if TEMP.THE_WITHIN /= 0 then
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_WITHIN - TEMP.THE_MET;
        else
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_PERIOD - TEMP.THE_MET;
        end if;
        EDL_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_UPPER, PRIORITY_QUEUE);
      end if;
      EDL_PRIORITY_QUEUES.REMOVE_BEST_FROM_PRIORITY_QUEUE(PRIORITY_QUEUE);
    else --* Scheduling Initial Set of Operators
      NEW_NODE.THE_START := STOP_TIME;
      TEMP := NEW_GRAPH.OP_RETURN(NEW_NODE.THE_OPERATOR);
      STOP_TIME := STOP_TIME + TEMP.THE_MET;
      NEW_NODE.THE_STOP := STOP_TIME;
      NEW_NODE.THE_INSTANCE := 1;
      EST := NEW_NODE.THE_START + TEMP.THE_PERIOD;
      if EST + TEMP.THE_MET <= H_B_LENGTH or else NEW_NODE.THE_START >= TEMP.THE_PERIOD then
        ADDL_NODE.THE_OPERATOR := NEW_NODE.THE_OPERATOR;
        ADDL_NODE.THE_LOWER := EST;
        ADDL_NODE.THE_INSTANCE := 1;
        if TEMP.THE_WITHIN /= 0 then
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_WITHIN - TEMP.THE_MET;
        else
          ADDL_NODE.THE_UPPER := EST + TEMP.THE_PERIOD - TEMP.THE_MET;
        end if;
        EDL_PRIORITY_QUEUES.INSERT_IN_PRIORITY_QUEUE(ADDL_NODE, ADDL_NODE.THE_UPPER, PRIORITY_QUEUE);
      end if;
      NODE_LIST.NEXT(T_SORT);
    end if;
    if NEW_NODE.THE_STOP > H_B_LENGTH then
      VALID_SCHEDULE := false;
    end if;
    SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, REV_AGENDA);
    NEW_NODE.THE_LOWER := 0;
    NEW_NODE.THE_UPPER := 0;
  end loop;
  SCHEDULE_INPUTS_LIST.LIST_REVERSE(REV_AGENDA, AGENDA);
  SCHEDULE_INPUTS_LIST.FREE_LIST(REV_AGENDA);
end EARLIEST_DEADLINE;

procedure EXHAUSTIVE_ENUMERATION ( TOP_SORT        : in NODE_LIST.LIST;
                                   AGENDA     : in out SCHEDULE_INPUTS_LIST.LIST;
                                   OP_COUNT        : in INTEGER;
                                   H_B_LENGTH    : in INTEGER;
                                   VALID_SCHEDULE: in out BOOLEAN) is

package int_io is new TEXT_IO.integer_io(integer);
use int_io;
```

```
TEMP                              : SCHEDULE_INPUTS_LIST.LIST;
COUNT                             : INTEGER := 0;

procedure TOP_SORTS (AGENDA               : in out SCHEDULE_INPUTS_LIST.LIST;
                     COUNT                : in INTEGER;
                     VALID_SCHEDULE       : in out BOOLEAN;
                     BLOCK_LENGTH         : in INTEGER) is

type VECTOR is array (1..COUNT) of INTEGER;
LOC                               : VECTOR;
type SCHEDULE_ARRAY is array (1..COUNT) of SCHEDULE_INPUTS;
P_ARRAY                           : SCHEDULE_ARRAY;
type TIME_RECORD is
  record
   OPERATOR                       : INTEGER;
   TIME_1                         : INTEGER;
   TIME_2                         : INTEGER;
end record;
type TIME_ARRAY is array (1..OP_COUNT+1) of TIME_RECORD;
START_TIME_ARRAY                  : TIME_ARRAY;

HOLD                              : SCHEDULE_INPUTS;
TEMP                              : SCHEDULE_INPUTS_LIST.LIST := AGENDA;
INDEX                             : INTEGER := 1;
INDEX_1                           : INTEGER := 1;
NODE_1                            : INTEGER;
NODE_2                            : INTEGER;
MET                               : INTEGER;
POSITION                          : INTEGER;
STOP_TIME                         : INTEGER;
HOLD_STOP_TIME                    : INTEGER;
START_TIME                        : INTEGER;
i                                 : INTEGER := COUNT;
INITIAL_START_TIME                : INTEGER;
LOWER_BOUND                       : INTEGER;
ADJUSTED                          : BOOLEAN := false;


begin
  while SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP) loop
    P_ARRAY(INDEX) := SCHEDULE_INPUTS_LIST.VALUE(TEMP);
    LOC(INDEX) := INDEX;
    INDEX := INDEX + 1;
    if SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_INSTANCE = 1 then
      START_TIME_ARRAY(INDEX_1).OPERATOR:= SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_OPERATOR;
      START_TIME_ARRAY(INDEX_1).TIME_1:= SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_START;
      START_TIME_ARRAY(INDEX_1).TIME_2:= SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_STOP;
      INDEX_1 := INDEX_1+1;
    end if;
    SCHEDULE_INPUTS_LIST.NEXT(TEMP);
   end loop;
  while i > 1 loop
    NODE_1 := P_ARRAY(LOC(i)).THE_OPERATOR;
    NODE_2 := P_ARRAY(LOC(i)-1).THE_OPERATOR;
    if not FRONT_END.NEW_GRAPH.IS_PARENT(NODE_2, NODE_1) then
      HOLD := P_ARRAY(LOC(i));
      if i > 2 then
        STOP_TIME := P_ARRAY(LOC(i)-1).THE_START;
      else
        STOP_TIME := 0;
      end if;
      P_ARRAY(LOC(i)) := P_ARRAY(LOC(i)-1);
      P_ARRAY(LOC(i)-1) := HOLD;
      STOP_TIME := 0;
      for i in 1..COUNT loop
         if P_ARRAY(i).THE_INSTANCE = 1 then
            for j in 1..OP_COUNT+1 loop
               if P_ARRAY(i).THE_OPERATOR = START_TIME_ARRAY(j).OPERATOR then
                  MET := START_TIME_ARRAY(j).TIME_2 - START_TIME_ARRAY(j).TIME_1;
                  P_ARRAY(i).THE_START := STOP_TIME;
                  START_TIME_ARRAY(j).TIME_1 := STOP_TIME;
                  STOP_TIME := STOP_TIME + MET;
                  START_TIME_ARRAY(j).TIME_2 := STOP_TIME;
                  P_ARRAY(i).THE_STOP := STOP_TIME;
                  exit;
            end if;
         end loop;
      else
         for l in 1..OP_COUNT+1 loop
            if P_ARRAY(i).THE_OPERATOR = START_TIME_ARRAY(l).OPERATOR then
               INITIAL_START_TIME := START_TIME_ARRAY(l).TIME_1;
               exit;
            end if;
         end loop;
         LOWER_BOUND := INITIAL_START_TIME + ((P_ARRAY(i).THE_INSTANCE-1) *
         NEW_GRAPH.OP_RETURN(P_ARRAY(i).THE_OPERATOR).THE_PERIOD);
         if LOWER_BOUND > STOP_TIME then
            START_TIME := LOWER_BOUND;
          else
            START_TIME := STOP_TIME;
         end if;
         MET := P_ARRAY(i).THE_STOP - P_ARRAY(i).THE_START;
         P_ARRAY(i).THE_START := START_TIME;
         STOP_TIME := START_TIME + MET;
         P_ARRAY(i).THE_STOP := STOP_TIME;
         P_ARRAY(i).THE_LOWER := LOWER_BOUND;
      end if;
   end loop;
   if P_ARRAY(COUNT).THE_STOP <= BLOCK_LENGTH then
      VALID_SCHEDULE := true;
      exit;
   end if;
   LOC(i) := LOC(i)-1;
   i := COUNT;
else
   if LOC(i) /= i then
      HOLD := P_ARRAY(LOC(i));
      for j in LOC(i) ..i-1 loop
         P_ARRAY(j) := P_ARRAY(j+1);
      end loop;
      P_ARRAY(i) := HOLD;
      LOC(i) := i;
      STOP_TIME := 0;
      for i in 1..COUNT loop
         if P_ARRAY(i).THE_INSTANCE = 1 then
            for j in 1..OP_COUNT+1 loop
               if P_ARRAY(i).THE_OPERATOR = START_TIME_ARRAY(j).OPERATOR then
                  MET := START_TIME_ARRAY(j).TIME_2 - START_TIME_ARRAY(j).TIME_1;
                  P_ARRAY(i).THE_START := STOP_TIME;
                  START_TIME_ARRAY(j).TIME_1 := STOP_TIME;
                  STOP_TIME := STOP_TIME + MET;
```

```
              START_TIME_ARRAY(j).TIME_2 := STOP_TIME;
              P_ARRAY(i).THE_STOP := STOP_TIME;
              exit;
            end if;
          end loop;
        else
          for l in 1..OP_COUNT+1 loop
            if P_ARRAY(i).THE_OPERATOR = START_TIME_ARRAY(l).OPERATOR then
              INITIAL_START_TIME := START_TIME_ARRAY(l).TIME_1;
              exit;
            end if;
          end loop;
          LOWER_BOUND := INITIAL_START_TIME + ((P_ARRAY(i).THE_INSTANCE-1) *
          NEW_GRAPH.OP_RETURN(P_ARRAY(i).THE_OPERATOR).THE_PERIOD);
          if LOWER_BOUND > STOP_TIME then
            START_TIME := LOWER_BOUND;
          else
            START_TIME := STOP_TIME;
          end if;
          MET := P_ARRAY(i).THE_STOP - P_ARRAY(i).THE_START;
          P_ARRAY(i).THE_START := START_TIME;
          STOP_TIME := START_TIME + MET;
          P_ARRAY(i).THE_STOP := STOP_TIME;
          P_ARRAY(i).THE_LOWER := LOWER_BOUND;
        end if;
      end loop;
    end if;
    i := i-1;
  end if;
  end loop;
  SCHEDULE_INPUTS_LIST.FREE_LIST(AGENDA);
  for l in reverse 1..COUNT loop
    SCHEDULE_INPUTS_LIST.ADD(P_ARRAY(l), AGENDA);
  end loop;
end TOP_SORTS;


  begin
    EARLIEST_START(TOP_SORT,AGENDA,OP_COUNT,H_B_LENGTH, VALID_SCHEDULE);
    TEMP := AGENDA;
    while SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP) loop
      COUNT := COUNT + 1;
      SCHEDULE_INPUTS_LIST.NEXT(TEMP);
    end loop;
    TOP_SORTS(AGENDA, COUNT, VALID_SCHEDULE,H_B_LENGTH);
  end EXHAUSTIVE_ENUMERATION;



  procedure CREATE_STATIC_SCHEDULE (OPERATOR_LIST          : in NODE_LIST.LIST;
                          THE_SCHEDULE_INPUTS : in SCHEDULE_INPUTS_LIST.LIST;
                          HARMONIC_BLOCK_LENGTH : in INTEGER) is
  -- creates the static schedule output and prints to "ss.a" file.
    OP_LIST                  : NODE_LIST.LIST := OPERATOR_LIST;
    S                        : SCHEDULE_INPUTS_LIST.LIST := THE_SCHEDULE_INPUTS;
    SCHEDULE                 : TEXT_IO.FILE_TYPE;
    OUTPUT                   : TEXT_IO.FILE_MODE := TEXT_IO.OUT_FILE;
    COUNTER                  : INTEGER := 1;
    TEMPVAR                  : OPERATOR_ID;

  package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);
  use VALUE_IO;
  package F_IO is new TEXT_IO.FLOAT_IO(FLOAT);
  package INTEGERIO is new TEXT_IO.INTEGER_IO(INTEGER);
  use INTEGERIO;

  begin
    TEXT_IO.CREATE(SCHEDULE, OUTPUT, “ss.a”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with GLOBAL_DECLARATIONS; use GLOBAL_DECLARATIONS;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with DS_DEBUG_PKG; use DS_DEBUG_PKG;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with TL; use TL;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with DS_PACKAGE; use DS_PACKAGE;”);
    TEXT_IO.PUT(SCHEDULE, “with PRIORITY_DEFINITIONS; “);
    TEXT_IO.PUT_LINE (SCHEDULE, “use PRIORITY_DEFINITIONS;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with CALENDAR; use CALENDAR;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “with TEXT_IO; use TEXT_IO;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “procedure STATIC_SCHEDULE is”);
    NODE_LIST.NEXT(OP_LIST); --* Bypass dummy start node
    while NODE_LIST.NON_EMPTY(OP_LIST) loop
      TEXT_IO.SET_COL(SCHEDULE, 3);
      VARSTRING.PUT(SCHEDULE, NEW_GRAPH.OP_RETURN(NODE_LIST.VALUE(OP_LIST)).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, “_TIMING_ERROR : exception;”);
      NODE_LIST.NEXT(OP_LIST);
    end loop;

    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “task type SCHEDULE_TYPE is”);
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT_LINE(SCHEDULE, “pragma priority (STATIC_SCHEDULE_PRIORITY);”);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “end SCHEDULE_TYPE;”);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “for SCHEDULE_TYPE’STORAGE_SIZE use 200_000;”);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “SCHEDULE : SCHEDULE_TYPE;”);
    TEXT_IO.NEW_LINE(SCHEDULE);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “task body SCHEDULE_TYPE is”);
    TEXT_IO.PUT(SCHEDULE, “ PERIOD : duration := duration(“);
    F_IO.PUT(SCHEDULE, FLOAT(HARMONIC_BLOCK_LENGTH)/1000.0);
    TEXT_IO.PUT_LINE(SCHEDULE, “);”);
    S := THE_SCHEDULE_INPUTS;
    SCHEDULE_INPUTS_LIST.NEXT(S); --* Bypass dummy start node.
    while SCHEDULE_INPUTS_LIST.NON_EMPTY(S) loop
      TEXT_IO.SET_COL(SCHEDULE, 5);
      VARSTRING.PUT(SCHEDULE,
FRONT_END.NEW_GRAPH.OP_RETURN(SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATO
R).THE_OPERATOR_ID);
      TEXT_IO.PUT(SCHEDULE, “_STOP_TIME”);
      INTEGERIO.PUT(SCHEDULE, COUNTER,1);
      TEXT_IO.PUT(SCHEDULE, “ : duration := duration(“);
      F_IO.PUT(SCHEDULE,FLOAT(SCHEDULE_INPUTS_LIST.VALUE(S).THE_STOP)/1000.0);
      TEXT_IO.PUT_LINE(SCHEDULE, “);”);
       SCHEDULE_INPUTS_LIST.NEXT(S);
      COUNTER := COUNTER + 1;
    end loop;
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT_LINE(SCHEDULE, “SLACK_TIME : duration;”);
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT_LINE(SCHEDULE, “START_OF_PERIOD : time := clock;”);
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT_LINE(SCHEDULE, “CURRENT_TIME : duration;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “begin”);
    TEXT_IO.PUT_LINE(SCHEDULE, “ loop”);
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT(SCHEDULE, “begin”);

    S := THE_SCHEDULE_INPUTS;
    SCHEDULE_INPUTS_LIST.NEXT(S); --* Bypass dummy start node.
    COUNTER := 1;
    while SCHEDULE_INPUTS_LIST.NON_EMPTY(S) loop
      TEXT_IO.SET_COL(SCHEDULE, 7);
      VARSTRING.PUT(SCHEDULE,
FRONT_END.NEW_GRAPH.OP_RETURN(SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATO
R).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, “_DRIVER;”);
      TEXT_IO.SET_COL(SCHEDULE, 7);
      TEXT_IO.PUT(SCHEDULE, “SLACK_TIME := START_OF_PERIOD + “);
      VARSTRING.PUT(SCHEDULE,
FRONT_END.NEW_GRAPH.OP_RETURN(SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATO
R).THE_OPERATOR_ID);
      TEXT_IO.PUT(SCHEDULE, “_STOP_TIME”);
      INTEGERIO.PUT(SCHEDULE, COUNTER,1);
      TEXT_IO.PUT_LINE(SCHEDULE, “ - CLOCK;”);
      TEXT_IO.SET_COL(SCHEDULE, 7);
      TEXT_IO.PUT_LINE(SCHEDULE, “if SLACK_TIME >= 0.0 then”);
      TEXT_IO.SET_COL(SCHEDULE, 9);
      TEXT_IO.PUT_LINE(SCHEDULE, “delay (SLACK_TIME);”);
      TEXT_IO.SET_COL(SCHEDULE, 7);
      TEXT_IO.PUT_LINE(SCHEDULE, “else”);
      TEXT_IO.SET_COL(SCHEDULE, 9);
      TEXT_IO.PUT(SCHEDULE, “raise “);
      VARSTRING.PUT(SCHEDULE,
FRONT_END.NEW_GRAPH.OP_RETURN(SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATO
R).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, “_TIMING_ERROR;”);
      TEXT_IO.SET_COL(SCHEDULE, 7);
      TEXT_IO.PUT_LINE(SCHEDULE, “end if;”);
      TEMPVAR:=
FRONT_END.NEW_GRAPH.OP_RETURN(SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATO
R).THE_OPERATOR_ID;
      SCHEDULE_INPUTS_LIST.NEXT(S);
      if SCHEDULE_INPUTS_LIST.NON_EMPTY(S) then
      -- pointer is pointing to the next record after this.
        TEXT_IO.SET_COL(SCHEDULE, 7);
        TEXT_IO.PUT(SCHEDULE, “delay (START_OF_PERIOD + “);
        VARSTRING.PUT(SCHEDULE, TEMPVAR);
        TEXT_IO.PUT(SCHEDULE, “_STOP_TIME”);
        INTEGERIO.PUT(SCHEDULE, COUNTER,1);
        TEXT_IO.PUT_LINE(SCHEDULE, “ - CLOCK);”);
        TEXT_IO.NEW_LINE(SCHEDULE);
      end if;
      COUNTER := COUNTER + 1;
    end loop;

    TEXT_IO.SET_COL(SCHEDULE, 7);
    TEXT_IO.PUT_LINE(SCHEDULE,
            “START_OF_PERIOD := START_OF_PERIOD + PERIOD;”);
    TEXT_IO.SET_COL(SCHEDULE, 7);
    TEXT_IO.PUT_LINE(SCHEDULE, “delay (START_OF_PERIOD - clock);”);

    TEXT_IO.SET_COL(SCHEDULE, 7);
    TEXT_IO.PUT_LINE(SCHEDULE, “exception”);
    OP_LIST := OPERATOR_LIST;
    NODE_LIST.NEXT(OP_LIST); --* Bypass dummy start node
    COUNTER:= COUNTER - 1;
    while NODE_LIST.NON_EMPTY(OP_LIST) loop
      TEXT_IO.SET_COL(SCHEDULE, 9);
      TEXT_IO.PUT(SCHEDULE, “when “);
      VARSTRING.PUT(SCHEDULE,
NEW_GRAPH.OP_RETURN(NODE_LIST.VALUE(OP_LIST)).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, “_TIMING_ERROR =>”);
      TEXT_IO.SET_COL(SCHEDULE, 11);
      TEXT_IO.PUT(SCHEDULE, “PUT_LINE(““timing error from operator “);
      VARSTRING.PUT(SCHEDULE,
NEW_GRAPH.OP_RETURN(NODE_LIST.VALUE(OP_LIST)).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, ““““);”);
      TEXT_IO.PUT_LINE(SCHEDULE, “START_OF_PERIOD := clock;”);
      NODE_LIST.NEXT(OP_LIST);
      COUNTER:= COUNTER - 1;
    end loop;

    TEXT_IO.SET_COL(SCHEDULE, 7);
    TEXT_IO.PUT_LINE(SCHEDULE, “end;”);
    TEXT_IO.SET_COL(SCHEDULE, 5);
    TEXT_IO.PUT_LINE(SCHEDULE, “end loop;”);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “end SCHEDULE_TYPE;”);
    TEXT_IO.NEW_LINE(SCHEDULE);
    TEXT_IO.PUT_LINE(SCHEDULE, “begin”);
    TEXT_IO.SET_COL(SCHEDULE, 3);
    TEXT_IO.PUT_LINE(SCHEDULE, “null;”);
    TEXT_IO.PUT_LINE(SCHEDULE, “end STATIC_SCHEDULE;”);

  end CREATE_STATIC_SCHEDULE;

end SCHEDULER;
```

```
with DATA; use DATA;

package ANNEAL is

    procedure SIMULATED_ANNEAL (PRECEDENCE_LIST : in NODE_LIST.LIST;
                                AGENDA : in out SCHEDULE_INPUTS_LIST.LIST;
                                H_B_LENGTH : in INTEGER;
                                VALID_SCHEDULE : in out BOOLEAN);

 end ANNEAL;
```

```
with TEXT_IO;
with DIAGNOSTICS;
with RANDOM;
with MATH; --* Necessary for EXP function.
with DATA; use DATA;
with FRONT_END; use FRONT_END;

package body ANNEAL is

  package int_io is new TEXT_IO.integer_io(integer);--put in for debugging
  use int_io;
  package float_io is new TEXT_IO.float_io(float);--put in for debugging
  use float_io;

  ----------------------------------------------------------------------------------------
  --* The following code is a modification of the HARMONIC BLOCK WITH PRECEDENCE
  --* CONSTRAINTS scheduling algorithm developed and implemented by Kilic. It is
  --* intended to develop an initial solution.

   procedure CREATE_INTERVAL (THE_OPERATOR : in OPERATOR;
                              INPUT          : in out SCHEDULE_INPUTS;
                              OLD_LOWER      : in VALUE) is
     LOWER_BOUND : VALUE;

   function CALC_LOWER_BOUND return VALUE is
   begin
   -- since CREATE_INTERVAL function is used in both SCHEDULE_INITIAL_SET and
   -- SCHEDULE_REST_OF_BLOCK (OLD_LOWER /= 0) check is needed.In case of the
   -- operator is scheduled somewhere in its interval and (OLD_LOWER /= 0),
   -- this check guarantees that the periods will be consistent.
      if (OLD_LOWER /= 0) then --* Schedule subsequent instance of task
        LOWER_BOUND := OLD_LOWER;
      else --* Schedule first instance of task
        LOWER_BOUND := INPUT.THE_START;
      end if;
      return LOWER_BOUND;
   end CALC_LOWER_BOUND;

   function CALC_UPPER_BOUND return VALUE is
   begin
      if THE_OPERATOR.THE_WITHIN = 0 then
        return LOWER_BOUND + THE_OPERATOR.THE_PERIOD - THE_OPERATOR.THE_MET;
      -- if the operator has a WITHIN constraint, the upper bound of the
      -- interval is reduced.
      else
        return LOWER_BOUND + THE_OPERATOR.THE_WITHIN - THE_OPERATOR.THE_MET;
      end if;
   end CALC_UPPER_BOUND;
   begin --main CREATE_INTERVAL
     INPUT.THE_LOWER := CALC_LOWER_BOUND;
     INPUT.THE_UPPER := CALC_UPPER_BOUND;
   end CREATE_INTERVAL;
   ----------------------------------------------
   procedure SCHEDULE_INITIAL_SET (PRECEDENCE_LIST : in NODE_LIST.LIST;
                         THE_SCHEDULE_INPUTS : in out SCHEDULE_INPUTS_LIST.LIST;
                         HARMONIC_BLOCK_LENGTH : in INTEGER;
                         STOP_TIME : in out INTEGER) is
   V                        : NODE_LIST.LIST := PRECEDENCE_LIST;
   START_TIME               : INTEGER := 0;
   NEW_INPUT                : SCHEDULE_INPUTS;
   OLD_LOWER                : VALUE :=0;
   OP_NUM                   : INTEGER;
   TEMP                     : OPERATOR;

begin --SCHEDULE_INITIAL_SET
   SCHEDULE_INPUTS_LIST.EMPTY(THE_SCHEDULE_INPUTS);
   NEW_INPUT.THE_OPERATOR := 0; --* This Code schedules
   NEW_INPUT.THE_LOWER := HARMONIC_BLOCK_LENGTH +10;--* the first and only
   SCHEDULE_INPUTS_LIST.ADD (NEW_INPUT, THE_SCHEDULE_INPUTS);--* instance of the
   NODE_LIST.NEXT(V);--* dummy start node.
   while NODE_LIST.NON_EMPTY(V) loop
      OP_NUM := NODE_LIST.VALUE(V);
      TEMP := NEW_GRAPH.OP_RETURN(OP_NUM);
      NEW_INPUT.THE_OPERATOR := OP_NUM;
      NEW_INPUT.THE_START := START_TIME;
      STOP_TIME := START_TIME + TEMP.THE_MET;
      NEW_INPUT.THE_STOP := STOP_TIME;
      START_TIME := STOP_TIME;
      -- for every operator in SCHEDULE_INITIAL_SET, OLD_LOWER is zero. So we
      -- always send zero value to CREATE_INTERVAL.
      CREATE_INTERVAL(TEMP, NEW_INPUT, OLD_LOWER);
      SCHEDULE_INPUTS_LIST.ADD (NEW_INPUT, THE_SCHEDULE_INPUTS);
      NODE_LIST.NEXT(V);
   end loop;
end SCHEDULE_INITIAL_SET;
--------------------------------------------------
procedure SCHEDULE_REST_OF_BLOCK(PRECEDENCE_LIST:in NODE_LIST.LIST;
                            THE_SCHEDULE_INPUTS : in out SCHEDULE_INPUTS_LIST.LIST;
                            HARMONIC_BLOCK_LENGTH : in INTEGER;
                            STOP_TIME : in INTEGER) is

   V                        : NODE_LIST.LIST := PRECEDENCE_LIST;
   TEMP                     : SCHEDULE_INPUTS_LIST.LIST := THE_SCHEDULE_INPUTS;
   V_LIST,
   HEAD                     : NODE_LIST.LIST;
   P                        : SCHEDULE_INPUTS_LIST.LIST;
   S                        : SCHEDULE_INPUTS_LIST.LIST;
   T                        : SCHEDULE_INPUTS_LIST.LIST;
   START_TIME               : INTEGER := 0;
   TIME_STOP                : INTEGER := STOP_TIME;
   NEW_INPUT                : SCHEDULE_INPUTS;
   OLD_LOWER                : VALUE;
   OUTSIDE_BLOCK            : BOOLEAN := false;
   OP_NUM                   : INTEGER;
   TEMP_OP                  : OPERATOR;


begin
   NODE_LIST.DUPLICATE(PRECEDENCE_LIST, V_LIST);

   SCHEDULE_INPUTS_LIST.LIST_REVERSE(THE_SCHEDULE_INPUTS, P);
   T := P;

   loop
   while SCHEDULE_INPUTS_LIST.NON_EMPTY(P) loop
  --* Changed < to <= on 1 Apr 91 to correct flaw in scheduler
     OP_NUM := NODE_LIST.VALUE(V);
   TEMP_OP := NEW_GRAPH.OP_RETURN(OP_NUM);
   if (SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER
     + TEMP_OP.THE_PERIOD
       + TEMP_OP.THE_MET) <= HARMONIC_BLOCK_LENGTH then
     NEW_INPUT.THE_OPERATOR := OP_NUM;
   --* The following if statement determines the appropriate start time
   --* of an operator.
     if SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER
       + TEMP_OP.THE_PERIOD >= TIME_STOP then
       START_TIME := SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER + TEMP_OP.THE_PERIOD;
     else
       START_TIME := TIME_STOP;
     end if;
     NEW_INPUT.THE_START := START_TIME;
     NEW_INPUT.THE_STOP := START_TIME + TEMP_OP.THE_MET;
     TIME_STOP := NEW_INPUT.THE_STOP;
     OLD_LOWER := SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER + TEMP_OP.THE_PERIOD;
     CREATE_INTERVAL(TEMP_OP, NEW_INPUT, OLD_LOWER);
     NEW_INPUT.THE_INSTANCE := SCHEDULE_INPUTS_LIST.VALUE(P).THE_INSTANCE + 1;
     SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT, TEMP);
     SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT, S);
     NODE_LIST.NEXT(V);
     SCHEDULE_INPUTS_LIST.NEXT(P);
     else
     NODE_LIST.NEXT(V);
     NODE_LIST.REMOVE(OP_NUM, V_LIST);
     SCHEDULE_INPUTS_LIST.NEXT(P);
   end if;
  end loop;
   if SCHEDULE_INPUTS_LIST.NON_EMPTY(S) then
     SCHEDULE_INPUTS_LIST.FREE_LIST(T);
     SCHEDULE_INPUTS_LIST.LIST_REVERSE(S, P);
     SCHEDULE_INPUTS_LIST.FREE_LIST(S);
     T := P;
     V := V_LIST;
   else
     exit;
   end if;
  end loop;
  SCHEDULE_INPUTS_LIST.LIST_REVERSE(TEMP, THE_SCHEDULE_INPUTS);
  SCHEDULE_INPUTS_LIST.FREE_LIST(TEMP);
end SCHEDULE_REST_OF_BLOCK;



-----------------------------------------------------------------------------
--* All code beyond this point is utilized by the SIMULATED ANNEALING algorithm *--


procedure TEST_SCHEDULE ( AGENDA          : in SCHEDULE_INPUTS_LIST.LIST;
                          COST            : in out INTEGER;
                          BLOCK_LENGTH    : in INTEGER;
                          OUTSIDE_BLOCK   : in out BOOLEAN) is
--* This procedure finds the cost of a schedule by traversing through it.


  V                   : SCHEDULE_INPUTS_LIST.LIST := AGENDA;
  PREVIOUS            : SCHEDULE_INPUTS_LIST.LIST := null;
```

```
begin
  COST := 0;
  SCHEDULE_INPUTS_LIST.NEXT(V); --Bypass Dummy Start Node
  while SCHEDULE_INPUTS_LIST.NON_EMPTY(V) loop
  if SCHEDULE_INPUTS_LIST.VALUE(V).THE_START
     < SCHEDULE_INPUTS_LIST.VALUE(V).THE_LOWER then
     COST := COST + (SCHEDULE_INPUTS_LIST.VALUE(V).THE_LOWER
         - SCHEDULE_INPUTS_LIST.VALUE(V).THE_START);
  elsif SCHEDULE_INPUTS_LIST.VALUE(V).THE_START
        > SCHEDULE_INPUTS_LIST.VALUE(V).THE_UPPER then
     COST := COST + (SCHEDULE_INPUTS_LIST.VALUE(V).THE_START
         - SCHEDULE_INPUTS_LIST.VALUE(V).THE_UPPER);
  end if;
  PREVIOUS := V;
  SCHEDULE_INPUTS_LIST.NEXT(V);
  end loop;
  if SCHEDULE_INPUTS_LIST.VALUE(PREVIOUS).THE_STOP > BLOCK_LENGTH then
     OUTSIDE_BLOCK := true;--* Schedule exceeds harmonic block length Not acceptable
  end if;
end TEST_SCHEDULE;



procedure ADJUST_SCHEDULE(TEMP_AGENDA       : in out SCHEDULE_INPUTS_LIST.LIST;
                          PRECEDENCE_LIST : in out NODE_LIST.LIST;
                          H_B_LENGTH          : in INTEGER;
                          OUTSIDE_HARMONIC_BLOCK : in out BOOLEAN;
                          NEW_LIST                : in out BOOLEAN) is
--* This procedure developes a new schedule based on another schedule
   HOLD,
   ADJUST_POINT          : SCHEDULE_INPUTS_LIST.LIST; --* op that misses deadline
   V                     : SCHEDULE_INPUTS_LIST.LIST := TEMP_AGENDA;
                         --* Original Schedule
   PENALTY_COST,
   MET,
   START_TIME,
   STOP_TIME             : INTEGER := 0;
   NEW_INPUT             : SCHEDULE_INPUTS;
   MOVED                 : BOOLEAN := false;
   ADJUSTED              : BOOLEAN := false;
   REDO                  : BOOLEAN := false;

procedure ADJUST_PRECEDENCE (PRECEDENCE_LIST: in out NODE_LIST.LIST) is
--* Develop a new precedence list.

   OP_TO_BE_RESCHEDULED,
   TEMP_PARENTS,
   PARENTS,
   TEMP                          : NODE_LIST.LIST;
   NEW_LIST,
   ADJUSTABLE,
   ADJUSTED,
   FOUND_PARENT,
   CAN_GO_NO_FURTHER             : BOOLEAN := false;
   RESCHEDULED_OP                : INTEGER;
   MOVE_COUNT                    : INTEGER := 0;

begin
  while not NEW_LIST loop
     TEMP := PRECEDENCE_LIST;
     while NODE_LIST.NON_EMPTY(TEMP) loop --Move to tail of list
```

```
      OP_TO_BE_RESCHEDULED := TEMP;
      NODE_LIST.NEXT(TEMP);
    end loop;
    MOVE_COUNT := INTEGER(RANDOM.NEXT_NUMBER * FLOAT(DATA.OP_COUNT));
    while MOVE_COUNT > 1 loop
      NODE_LIST.PREVIOUS(OP_TO_BE_RESCHEDULED);
      MOVE_COUNT := MOVE_COUNT - 1;
    end loop;
    TEMP := OP_TO_BE_RESCHEDULED;
    NODE_LIST.PREVIOUS(TEMP);
    while not ADJUSTED loop
      if not NODE_LIST.NON_EMPTY(TEMP) then
        exit; --* Cannot reschedule first op in list.
      end if;
      while NODE_LIST.NON_EMPTY(TEMP) loop
      if not NEW_GRAPH.IS_PARENT(NODE_LIST.VALUE(TEMP),
        NODE_LIST.VALUE(OP_TO_BE_RESCHEDULED)) then
        ADJUSTABLE := true;
        NODE_LIST.PREVIOUS(TEMP);
      else
        exit;
      end if;
    end loop;
    if ADJUSTABLE then
      RESCHEDULED_OP := NODE_LIST.VALUE(OP_TO_BE_RESCHEDULED);
      NODE_LIST.REMOVE(RESCHEDULED_OP, PRECEDENCE_LIST);
      NODE_LIST.INSERT_NEXT(RESCHEDULED_OP, TEMP);
      ADJUSTED := true;
      NEW_LIST := true;
    else
      NODE_LIST.PREVIOUS(OP_TO_BE_RESCHEDULED);
      TEMP := OP_TO_BE_RESCHEDULED;
      NODE_LIST.PREVIOUS(TEMP);
    end if;
   end loop;
  end loop;

  end ADJUST_PRECEDENCE;

  ------------------------------------------------------------------------------------

  begin --* MAIN Adjust Schedule procedure
  --* This first loop traverse thru a copy of the agenda to find the first instance of an
  --* operator that misses its deadline the schedule will be adjusted from this point.
   while SCHEDULE_INPUTS_LIST.NON_EMPTY(V) loop
    if SCHEDULE_INPUTS_LIST.VALUE(V).THE_START
        > SCHEDULE_INPUTS_LIST.VALUE(V).THE_UPPER then
      ADJUST_POINT := V;
      exit;
    end if;
    SCHEDULE_INPUTS_LIST.NEXT(V);
   end loop;
   while not ADJUSTED loop
      if not SCHEDULE_INPUTS_LIST.NON_EMPTY(V) or REDO then
  --* At this point all operators meet their deadlines but the schedule exceeds the
  --* harmonic block length. The initial set of ops must be adjusted
      ADJUST_PRECEDENCE(PRECEDENCE_LIST);
      SCHEDULE_INPUTS_LIST.FREE_LIST(TEMP_AGENDA);
      SCHEDULE_INITIAL_SET(PRECEDENCE_LIST,TEMP_AGENDA,H_B_LENGTH,
STOP_TIME);
```

```
         SCHEDULE_REST_OF_BLOCK(PRECEDENCE_LIST,TEMP_AGENDA,H_B_LENGTH,
STOP_TIME);
         NEW_LIST := true;
         ADJUSTED := true;
       else
  --* The following if statement finds the point in the original AGENDA where we can begin
  --* to reschedule operators. It does so in reverse order from the point that the first
  --* operator missed its deadline back to the start point of the schedule. Each
  --* operator's start time and child relationships are checked to see if the operator
  --* that miseed its deadline (ADJUST POINT) can start prior to this operator.
         SCHEDULE_INPUTS_LIST.PREVIOUS(V);
         HOLD := V;
         while SCHEDULE_INPUTS_LIST.NON_EMPTY(V) loop
            if SCHEDULE_INPUTS_LIST.VALUE(V).THE_START
            > SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_LOWER
        and not NEW_GRAPH.IS_PARENT(SCHEDULE_INPUTS_LIST.VALUE(V).THE_OPERATOR,
                 SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_OPERATOR) then
              SCHEDULE_INPUTS_LIST.PREVIOUS(V);
              MOVED := true;
            else
              STOP_TIME := SCHEDULE_INPUTS_LIST.VALUE(V).THE_STOP;
              exit;
            end if;
          end loop;
          if MOVED then
            NEW_INPUT.THE_OPERATOR :=
SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_OPERATOR;
              if SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_LOWER > STOP_TIME
then
                 START_TIME := SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_LOWER;
              else
                 START_TIME := STOP_TIME;
              end if;
             NEW_INPUT.THE_START := START_TIME;
             MET:= SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_STOP
             - SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_START;
             STOP_TIME := START_TIME + MET;
             NEW_INPUT.THE_STOP := STOP_TIME;
             NEW_INPUT.THE_LOWER :=
SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_LOWER;
            --* These should stay the same
             NEW_INPUT.THE_INSTANCE :=
SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_INSTANCE;
             NEW_INPUT.THE_UPPER :=
SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT).THE_UPPER;
             SCHEDULE_INPUTS_LIST.INSERT_NEXT(NEW_INPUT, V);

SCHEDULE_INPUTS_LIST.REMOVE(SCHEDULE_INPUTS_LIST.VALUE(ADJUST_POINT),
TEMP_AGENDA);
             SCHEDULE_INPUTS_LIST.NEXT(V);
             ADJUSTED := true;
              while SCHEDULE_INPUTS_LIST.NON_EMPTY(V) loop
                 if SCHEDULE_INPUTS_LIST.VALUE(V).THE_LOWER <= STOP_TIME or
                   SCHEDULE_INPUTS_LIST.VALUE(V).THE_START < STOP_TIME then
                     if SCHEDULE_INPUTS_LIST.VALUE(V).THE_START > STOP_TIME then
                        exit;
                      end if;
                     NEW_INPUT.THE_OPERATOR := SCHEDULE_INPUTS_LIST.VALUE(V).THE_OPERATOR;
                     START_TIME := STOP_TIME;
                     NEW_INPUT.THE_START := START_TIME;
                     MET:= SCHEDULE_INPUTS_LIST.VALUE(V).THE_STOP
```

```
                    - SCHEDULE_INPUTS_LIST.VALUE(V).THE_START;
                    STOP_TIME := START_TIME + MET;
                    NEW_INPUT.THE_STOP := STOP_TIME;
                    NEW_INPUT.THE_LOWER :=
SCHEDULE_INPUTS_LIST.VALUE(V).THE_LOWER;
                    NEW_INPUT.THE_UPPER :=
SCHEDULE_INPUTS_LIST.VALUE(V).THE_UPPER;
                    SCHEDULE_INPUTS_LIST.REPLACE_ITEM(NEW_INPUT, V);
                   end if;
                  SCHEDULE_INPUTS_LIST.NEXT(V);
                end loop;
              end if;
            end if;
            if not ADJUSTED then
              REDO := true;
            end if;
        end loop;
  end ADJUST_SCHEDULE;

   procedure ANNEAL_PROCESS (H_B_LENGTH        : in INTEGER;
                             AGENDA              : in out SCHEDULE_INPUTS_LIST.LIST;
                             SOLUTION_FOUND   : in out BOOLEAN;
                             PENALTY_COST       : in out INTEGER;
                             PRECEDENCE_LIST  : in out NODE_LIST.LIST;
                             OUTSIDE_HARMONIC_BLOCK : in out BOOLEAN) is


     SCRATCH_AGENDA,
     BEST_AGENDA,
     TEMP_AGENDA                          : SCHEDULE_INPUTS_LIST.LIST;

     TEMPERATURE                          : FLOAT;
     BEST_COST,
     TEMP_COST                            : INTEGER := 0;
     TRIAL_NUM                            : INTEGER := 100;
     ACCEPT_NUM                           : INTEGER := 25;
     STOP_TIME,
     TRIAL_COUNT,
     ACCEPT_COUNT                         : INTEGER := 0;
     COOLING_FACTOR                       : FLOAT := 0.95;
     FREEZE                               : FLOAT := 1.0;
     NEW_PREC_LIST                        : BOOLEAN := false;

   function ANNEAL_FUNCTION (COST_1          : in INTEGER;
                             COST_2       : in INTEGER;
                             CURRENT_TEMPERATURE : in FLOAT) return FLOAT is

     DELTA_C                  : FLOAT;

   begin
     DELTA_C := (FLOAT(COST_1 - COST_2)/CURRENT_TEMPERATURE);
     if DELTA_C <= 15.0 then
       return MATH.EXP(-DELTA_C);
     else
       return 0.0;
     end if;
   end ANNEAL_FUNCTION;

   begin
    SCHEDULE_INPUTS_LIST.DUPLICATE(AGENDA, BEST_AGENDA);
    BEST_COST := PENALTY_COST;
    TEMPERATURE := 2.0 * FLOAT(PENALTY_COST);
    CHEDULE_INPUTS_LIST.DUPLICATE(AGENDA, TEMP_AGENDA);
    while not SOLUTION_FOUND and TEMPERATURE > FREEZE loop
      while TRIAL_COUNT < TRIAL_NUM and ACCEPT_COUNT < ACCEPT_NUM loop
        ADJUST_SCHEDULE(TEMP_AGENDA,PRECEDENCE_LIST,H_B_LENGTH,OUTSIDE_HARMONIC_BLOCK,
NEW_PREC_LIST);
        OUTSIDE_HARMONIC_BLOCK := false;
        TEST_SCHEDULE(TEMP_AGENDA,TEMP_COST,H_B_LENGTH,OUTSIDE_HARMONIC_BLOCK);
         if TEMP_COST <= PENALTY_COST or else RANDOM.NEXT_NUMBER
            < ANNEAL_FUNCTION(TEMP_COST, PENALTY_COST, TEMPERATURE) then
          if TEMP_COST < BEST_COST then
            BEST_COST := TEMP_COST;
            SCHEDULE_INPUTS_LIST.COPY_LIST(TEMP_AGENDA, BEST_AGENDA);
          end if;
          PENALTY_COST := TEMP_COST;
          SCRATCH_AGENDA := AGENDA;
          AGENDA := TEMP_AGENDA;
          TEMP_AGENDA := SCRATCH_AGENDA;
          ACCEPT_COUNT := ACCEPT_COUNT + 1;
        elsif NEW_PREC_LIST then
          SCRATCH_AGENDA := AGENDA;
          AGENDA := TEMP_AGENDA;
          TEMP_AGENDA := SCRATCH_AGENDA;
          PENALTY_COST := TEMP_COST;
          NEW_PREC_LIST := false;
          if TEMP_COST < BEST_COST then
            BEST_COST := TEMP_COST;
            SCHEDULE_INPUTS_LIST.COPY_LIST(TEMP_AGENDA, BEST_AGENDA);
          end if;
        end if;
        SCRATCH_AGENDA := null;
        TRIAL_COUNT := TRIAL_COUNT + 1;
        if PENALTY_COST <= 0 and not OUTSIDE_HARMONIC_BLOCK then
          SOLUTION_FOUND := true;
          exit;
        else
          SCHEDULE_INPUTS_LIST.COPY_LIST(AGENDA, TEMP_AGENDA);
        end if;
      end loop;
      ACCEPT_COUNT := 0;
      TRIAL_COUNT := 0;
      TEMPERATURE := TEMPERATURE * COOLING_FACTOR;
    end loop;
    if not SOLUTION_FOUND then
      AGENDA := BEST_AGENDA;
      PENALTY_COST := BEST_COST;
    end if;


  end ANNEAL_PROCESS;


  procedure SIMULATED_ANNEAL (PRECEDENCE_LIST : in NODE_LIST.LIST;
                              AGENDA          : in out SCHEDULE_INPUTS_LIST.LIST;
                              H_B_LENGTH      : in INTEGER;
                              VALID_SCHEDULE : in out BOOLEAN) is

    PENALTY_COST,
    TEMP_COST,
    STOP_TIME                         : INTEGER := 0; --* (MAR 91)
```

```
     ANNEAL                              : BOOLEAN := false;
     WORKING_PRECEDENCE_LIST             : NODE_LIST.LIST;
     OUTSIDE_HARMONIC_BLOCK              : BOOLEAN := false;
     A_AGENDA                            : SCHEDULE_INPUTS_LIST.LIST;
     BLANK                               : SCHEDULE_INPUTS;

   begin
     NODE_LIST.DUPLICATE(PRECEDENCE_LIST,WORKING_PRECEDENCE_LIST);
     SCHEDULE_INITIAL_SET(PRECEDENCE_LIST,AGENDA,H_B_LENGTH, STOP_TIME);
     SCHEDULE_REST_OF_BLOCK(PRECEDENCE_LIST,AGENDA,H_B_LENGTH, STOP_TIME);
     ANNEAL := false;

TEST_SCHEDULE(AGENDA,PENALTY_COST,H_B_LENGTH,OUTSIDE_HARMONIC_BLOCK);
     if PENALTY_COST > 0 or OUTSIDE_HARMONIC_BLOCK then --* Then Aneealing is required.
       OUTSIDE_HARMONIC_BLOCK := false;
       ANNEAL := true;
       RANDOM.INITIALIZE(2*DATA.OP_COUNT+1); --* Initialize Random Number Generator
       --* with an odd number.
     else
       VALID_SCHEDULE := true;
     end if;
     if ANNEAL then
       ANNEAL_PROCESS(H_B_LENGTH,AGENDA, VALID_SCHEDULE,PENALTY_COST,
       WORKING_PRECEDENCE_LIST,OUTSIDE_HARMONIC_BLOCK);
     end if;
   end SIMULATED_ANNEAL;


 end ANNEAL;
```

REFERENCES

Anderson, E.S., *Functional Specifications For Generic C^3I Station*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1990.

Bra, F. R., "Scheduling with Earliest Start and Due Date Constraints," *Naval Research Logistic Quarterly*, v. 18, n. 4, December 1971.

Cervantes, J.J., *An Optimal Static Scheduling Algorithm For Hard Real-Time Systems Specified in a Prototyping Language*, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1989.

Coskun, V. and Kesoglu, *A Software Prototype For A Command, Control, Communications, and Intelligence (C^3I) Workstation*, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1990.

Fan, B.H., *Evaluation of Some Scheduling Algorithms For Hard Real-time Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

Flannery, B.P. and others, *Numerical Recipes in C, The Art of Scientific Computing*, pp. 343- 352, Cambridge University Press, 1984.

Janson, D.M., *A Static Scheduler For the Computer Aided Prototyping System: An Implementation Guide*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1988.

Johnson, D.S. and others, "Optimization by Simulated Annealing: An Experimental Evaluation, Part I (Graph Partioning)," *Operations Research*, v. 37 pp. 865-892, November-December 1989.

Johnson, D.S. and others, "Optimization by Simulated Annealing: An Experimental Evaluation, Part II (Graph Coloring and Number Partioning)," *Operations Research*, to appear.

Kilic, M., *Static Schedulers For Embedded Real-time Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1989.

LuQi, *Rapid Prototyping For Large Software Systems Design*, Ph.D. Dissertation, University of Minnesota, Duluth, Minnesota, May 1986.

Mok, A., "A Graph Based Computational Model for Real-Time Systems", *Proceedings of the IEEE International Conference on Parallel Processing,* Pennsylvanian State University, 1985.

Otten, R.H. and Van Ginneken, L.P., *The Annealing Algorithm,* Kluwer Academic Publishers, 1989.

Shing, Man Tak, " Efficient Scheduling Algorithms for Rapid Prototyping of Hard Real-Time Systems", paper, Naval Postgraduate School, Monterey, California, May 1991.

Zdrzalka, S. " Scheduling Jobs On A Single Machine With Periodic Release Date/Deadline Intervals", *European Journal of Operations Research,* v.40, pp. 243-251, 1989.

INITIAL DISTRIBUTION

1. Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

2. Dudley Knox Library 2
Code 52
Naval Postgraduate School
Monterey, CA 93943

3. Computer Science Department 2
Code CS
Naval Postgraduate School
Monterey, CA 93943

4. Office of the Assistant Secretary of the Navy 1
Research Development and Acquisition
Department of the Navy
Attn: Mr. Gerald A. Cann
Washington, DC 20380-1000

5. Office of the Chief of Naval Operations 1
OP-094
Department of the Navy
Attn: VADM J. O. Tuttle, USN
Washington, DC 20301-3040

6. Director of Defense Information 1
Office of the Assistant Secretary of Defense
(Command, Control, Communications, & Intelligence)
Attn: Mr. Paul Strassmann
Washington, DC 20301-0208

7. Center for Naval Analysis...1
4401 Ford Avenue
Alexandria, VA 22302-0268

8. Director of Research Administration 1
Attn: Prof. Howard
Code 08Hk
Naval Postgraduate School
Monterey, CA 93943

9. Chairman, Code CS...1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

10. Prof. Luqi, Code CSLq...10
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

11. Chief of Naval Research..1
800 N. Quincy Street
Arlington, VA 22217

12. Director, Ada Joint Program Office...1
OUSDRE (R&AT)
Room 3E114, The Pentagon
Attn: Dr. John P. Solomond
Washington, DC 20301-0208

13. Carnegie Mellon University...1
Software Engineering Institute
Department of Computer Science
Attn: Dr. Lui Sha
Pittsburgh, PA 15260

14. Office of Naval Technology (ONT)...1
Code 227
Attn: Dr. Elizabeth Wald
800 N. Quincy St.
Arlington, VA 22217-5000

15. Defense Advanced Research Projects Agency (DARPA)..................................1
Integrated Strategic Technology Office (ISTO)
Attn: Dr. B. Boehm
1400 Wilson Boulevard
Arlington, VA 22209-2308

16. Defense Advanced Research Projects Agency (DARPA)..................................1
ISTO
1400 Wilson Boulevard
Attn: LCol Eric Mattala
Arlington, VA 2209-2308

17. Defense Advanced Research Projects Agency (DARPA)..........................1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308

18. Attn: Dr. Charles Harland..1
Computer Science
Department of the Air Force
Bolling Air Force Base, DC 20332-6448

19. Chief of Naval Operations...1
Attn: Dr. R. M. Carroll (OP-01B2)
Washington, DC 20350

20. Dr. Amiram Yehudai...1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, ISRAEL 69978

21. Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, CA 90292-6695

22. Dr. Ted Lewis...1
Oregon State University
Computer Science Department
Corvallis, OR 97331

23. International Software Systems Inc...1
12710 Research Boulevard, Suite 301
Attn: Dr. R. T. Yeh
Austin, TX 78759

24. Kestrel Institute 1
Attn: Dr. C. Green
1801 Page Mill Road
Palo Alto, CA 94304

25. National Science Foundation...1
Division of Computer and Computation Research
Attn: K. C. Tai
Washington, DC 20550

26. Commander Space and Naval Warfare Systems Command........................1
SPAWAR 3212
Department of the Navy
Attn: Cdr M. Romeo
Washington, DC 20363-5100

27. Naval Ocean Systems Center 1
Attn: Linwood Sutton, Code 423
San Diego, CA 92152-5000

28. Naval Ocean Systems Center 1
Attn: Les Anderson, Code 413
San Diego, CA 92152-5000

29. Commander, Naval Sea Systems Command (PMS-4123H)..........................1
Attn: William Wilder
Washington, DC 20380-1000

30. New Jersey Institute of Technology...1
Computer Science Department
Attn: Dr. Peter Ng
Newark, NJ 07102

31. Office of Naval Research...1
Computer Science Division, Code 1133
Attn: Dr. A. M. Van Tilborg
800 N. Quincy Street
Arlington, VA 22217-5000

32. Office of Naval Research...1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, VA 22217-5000

33. Oregon Graduate Center 1
Portland (Beaverton)
Attn: Dr. R. Kieburtz
Portland, OR 97005

34. Santa Clara University..1
Department of Electrical Engineering and
Computer Science
Attn: Dr. M. Ketabchi
Santa Clara, CA 95053

35. Software Group, MCC 1
9430 Research Boulevard
Attn: Dr. L. Belady
Austin, TX 78759

36. University of California at Berkeley..1
Department of Electrical Engineering and
Computer Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
Berkeley, CA 90024

37. University of California at Irvine...1
Department of Computer and Information Science
Attn: Dr. Nancy Leveson
Irvine, CA 92717

38. Chief of Naval Operations..1
Attn: Dr. Earl Chavis (OP-16T)
Washington, DC 20350

39. Office of the Chief of Naval Operations 1
Attn: Dr. John Davis (OP-094H)
Washington, DC 20350-2000

40. University of Illinois...1
Department of Computer Science
Attn: Dr. Jane W. S. Liu
Urbana Champaign, IL 61801

41. University of Maryland..1
College of Business Management
Tydings Hall, Room 0137
Attn: Dr. Alan Hevner
College Park, MD 20742

42. University of Maryland..1
Computer Science Department
Attn: Dr. N. Roussapoulos
College Park, MD 20742

43. University of Massachusetts...1
Department of Computer and Information Science
Attn: Dr. John A. Stankovic
Amherst, MA 01003

44. University of Minnesota..1
Computer Science Department
136 Lind Hall
207 Church Street SE
Attn: Dr. Slagle
Minneapolis, MN 55455

45. University of Texas at Austin...1
Computer Science Department
Attn: Dr. Al Mok
Austin, TX 78712

46. Commander, Naval Surface Warfare Center, ...1
Code U-33
Attn: Dr. Philip Hwang
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

47. Attn: George Sumiall...1
US Army Headquarters
CECOM
AMSEL-RD-SE-AST-SE
Fort Monmouth, NJ 07703-5000

48. Attn: Joel Trimble..1
1211 South Fern Street, C107
Arlington, VA 22202

49. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P. O. Box 12211
Research Triangle Park, NC 27709-2211

50. George Mason University...1
Computer Science Department
Attn: Dr. David Rine
Fairfax, VA 22030-4444

51. Hewlett Packard Research Laboratory 1
Mail Stop 321
1501 Page Mill Road
Attn: Dr. Martin Griss
Palo Alto, CA 94304

52. Carnegie Mellon University..1
SEI
Attn: Dr. Mario Barbacci
Pittsburgh, PA 15213

53. Persistent Data Systems..1
75 W. Chapel Ridge Road
Attn: Mr. John Nester
Pittsburgh, PA 15238

54. University of Pittsburgh 1
Department of Computer Science
Attn: Dr. Alfs Berztiss
Pittsburgh, PA 15260

55. Prof. Shing, Code CSSh...1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Thesis
L575 Levine
c.1 An efficient heuristic
scheduler for hard
real-time systems.